$2.95 in U.S.A.

FEATURING ORIGINAL PROGRAMS AND EXERCISES

UNDERSTANDING APPLE® GRAPHICS

Create Beautiful Graphics With These Easy-To-Follow Instructions

An Alfred Handy Guide

050N DEPT 134
PRICE 2.95
M 31 TM 1359

by Richard Peddicord

PLATE I. A color-bar test pattern (see page 12).

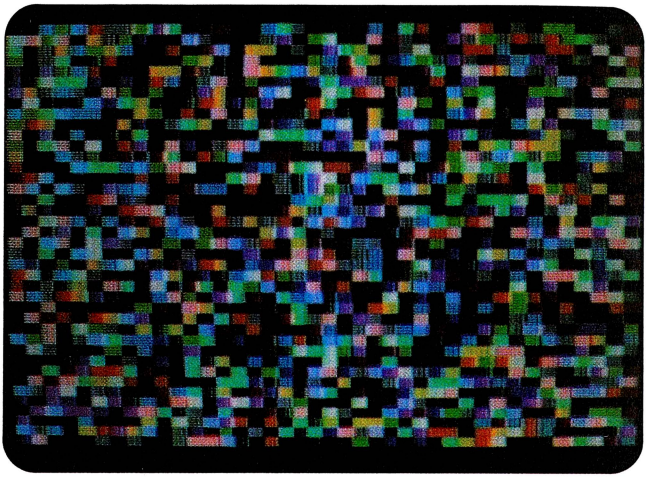


PLATE II. Kaleidoscope (see page 24).

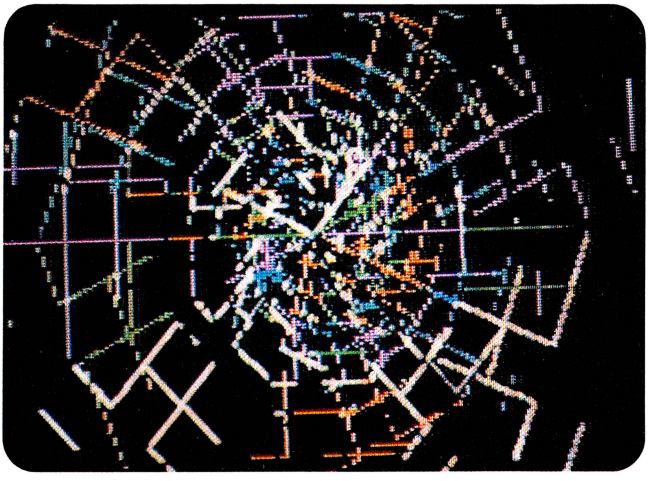


PLATE III. Random lines (see page 33).

UNDERSTANDING APPLE® GRAPHICS

by Richard G. Peddicord

AN ALFRED HANDY GUIDE

Computer Series Editor:
George Ledin Jr.

1. About This Handy Guide 3
2. About Your Apple 4
3. Low-Resolution Graphics 8
4. A Color-Bar Generator 12
5. Flag Designs 15
6. How To Make a Kaleidoscope 20
7. Low-Resolution Lines 27
8. High-Resolution Graphics 32
9. How To Draw a Circle 36
10. Creating Shapes 41
11. Using Shapes 45
12. Tic-Tac-Toe 50
13. Where To Go from Here 54

ALFRED PUBLISHING CO., INC.
SHERMAN OAKS, CA 91403

Acknowledgments

The author and the Series Editor thank Dick Merwin for providing solutions to the exercises, and Dick Henke for his useful suggestions.

10 9 8 7 6 5 4 3 2 1

Copyright © 1983 by Alfred Publishing Co. Inc.
Printed in the United States of America.
All rights reserved. No part of this book shall be reproduced or transmitted in any form by any means, electronic or mechanical, including photocopying, recording, or by any information or retrieval system without written permission of the publisher.

Alfred Publishing Co., Inc.
15335 Morrison St.
PO Box 5964
Sherman Oaks, CA 91413

Library of Congress Cataloging in Publication Data

Peddicord, Richard G.
Understanding Apple graphics.

(An Alfred handy guide)
Bibliography: p.
Includes index.
1. Computer graphics. 2. Apple //
(Computer)MProgramming. I. Title.
T385.P4 1983 001.64'43 83-15148
ISBN 0-88284-250-1

1. ABOUT THIS HANDY GUIDE

This Alfred Handy Guide is written for beginning students of computer graphics who have access to an Apple // computer. The Apple // is an excellent machine for color graphics, as it has two separate high-resolution screens and a shape-drawing system that includes rotation and scaling. All of the exercises in this book can be done also on the Apple //+ or Apple //e.

If you have no prior experience with BASIC we suggest you obtain the companion Handy Guide to this one, *Understanding Apple BASIC*, so that you can refer to it as needed. The bibliography at the back of this Handy Guide also lists several good BASIC textbooks.

In this Handy Guide you will learn those elementary concepts of computer graphics that are required to program in Applesoft. A glossary at the back will assist you with unfamiliar words.

You must learn some analytic geometry, algebra, and trigonometry in order to program successful graphic displays in Applesoft BASIC. These tools are required for drawing even the most primitive of graphic objects, lines and circles.

The shape-drawing feature of Applesoft permits you to define the outline of a shape by means of tiny vectors. Once defined and placed into memory, the shape can be scaled larger or smaller, and rotated in small steps. You will learn the necessary techniques to do this, which include conversion of small binary numbers to hexadecimals.

We have included exercises at the end of each section so that you will have projects to work on. There is no way to become good at something unless you practice a lot. Be sure to try the exercises, which are fun and challenging.

The Apple // was designed for everyone to enjoy. You should be able to learn about your Apple from this Handy Guide and the process should be enjoyable. If possible, find someone to assist you when you need help, and by all means spend lots of time on a machine.

2. ABOUT YOUR APPLE

The Apple // computer, made by Apple Computer Inc. of Cupertino, California, is one of the most popular microcomputers of all time; indeed, it sets a standard by which other microcomputers are presently judged. A great deal of software and hardware are available for this particular microcomputer system, and its Applesoft BASIC has become one of the most flexible BASICs in operation today.

The Apple // features a high-resolution color screen of 280 columns and 196 rows, with six different colors (black, white, orange, pink, blue, and green) available for each *pixel*, or picture element. This is about the same resolution and color range as the personal game computers on the market today. It follows that you can program games on your Apple //. It is not possible to attain the same speed as most commercial programs, however, because you will be working in Applesoft BASIC, and commercial programs are written in Assembly language.

The low-resolution screen on the Apple // has 40 columns and 40 rows. Each pixel can be one of 16 different colors from black to white. The reason why you have more choice of color with the lower resolution has to do with the way the color TV signal is generated, and how the color image is stored.

The configuration that has proven to be most cost-effective and useful consists of an Apple // computer with one disk drive connected to a TV set via an RF modulator. The RF modulator converts the Apple's TV signal into the same kind of signal that the television receiver is expecting. Most RF modulators for computers generate an output signal that goes to channel 3 or 4.

The first thing you must do is to check the connections on your system before turning on the power. Your Apple // computer has a power cord that plugs into the back (see Figure 2.1). Plug this into a *grounded* 115-volt AC circuit. If you try to use an ungrounded circuit, the manufacturer's warranty is void, for good reason.

Coming out of the Apple will be what looks like a hi-fi cable, and indeed it has the same kind of plug on the end, a so-called RCA Phono plug. This plugs into a little metal box that attaches to your TV set antenna, and to your outside TV antenna. There will be a sliding switch on the box that switches between the computer and the antenna. This arrangement permits the family color TV set to be used as your Apple's output device.

You may or may not have a disk drive connected to your Apple. If one is connected, so much the better. If one is not connected, do not attempt to connect one using only this Handy Guide. You must use the instructions supplied with the drive. You can, however, hook up your Apple // to a TV set and turn on the power to

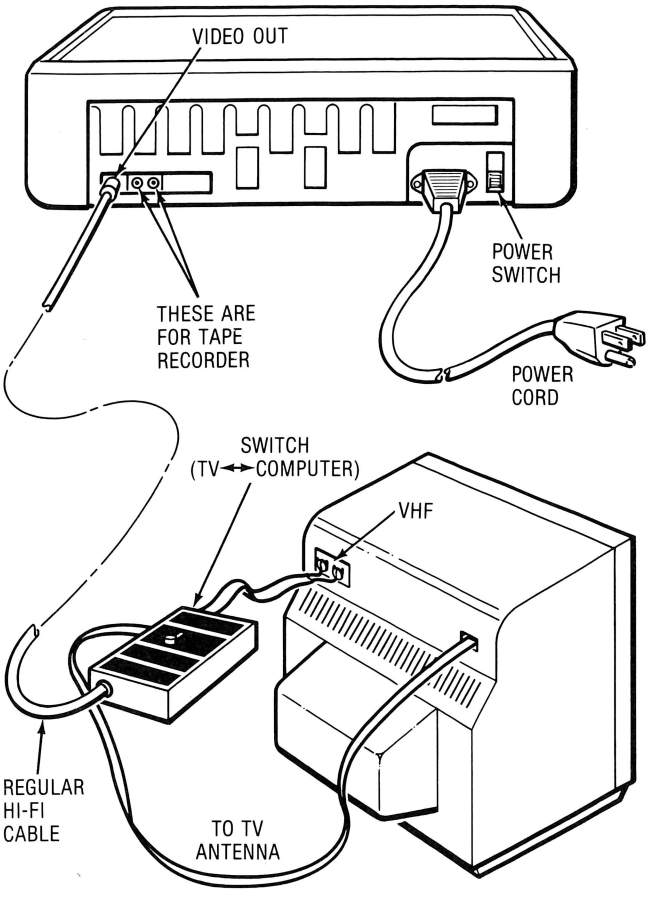


Figure 2.1 Hooking up your Apple // computer.

both. The Apple's power switch is located in the back on the far left. It is a rocker switch; push the top part in to turn the power on. The power light in the lower left corner of the keyboard will come on.

A diagram of the Apple // keyboard is shown in Figure 2.2. Your first step is to try out a few of the essential keys. The RESET key is most useful. Try it. You will hear a bleep from the Apple's speaker and the flashing square cursor will be immediately to the right of the Applesoft prompt character,]. Whatever program you were working on will still be in memory, and whatever shapes you had entered will still be there. In other words, no real harm can come to you if you press RESET.

The SHIFT key works the same as on a typewriter. Hold it while you strike a key; the upper-case version of the character will be sent to the central processing unit.

In Applesoft you enter one line at a time and then press RETURN. Until you press RETURN the machine allows you to change whatever you have typed in. Therefore you can make all the corrections you want to a line before entering it. The *arrow* keys (on the right side of the keyboard) are extremely useful in making corrections, as they move the cursor across the text without changing it. Only when you press another key will the indicated change be made. You should make full use of these keys.

If you want to move the cursor over many characters, you can hold down the repeat key (REPT) while

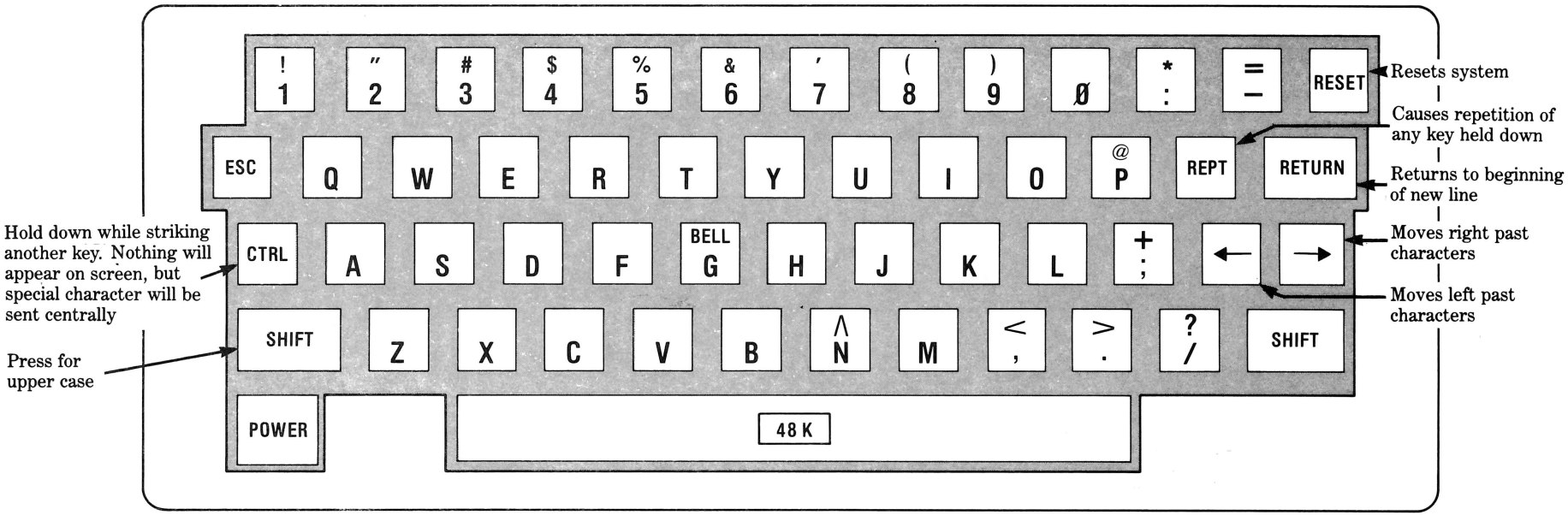


Figure 2.2 The Apple // keyboard.

you hold down one of the arrow keys. This same REPT key will cause any character to be repeated.

Some commands that you should know:

HOME	Clears text and moves cursor to top left
DEL	Put beginning line number after DEL, then a comma, then the ending line number. Every line from beginning to end will be DELeted from the program.
LIST	Lists your program.
RUN	Runs your program.
PR#1	Enables printer in slot 1.
PR#6	Enables disk drive in slot 6.
PR#0	Disables printer.

Exercise

Play follow-the-leader with us. Simply do each thing listed below as many times as indicated:

a. Press RESET 3 times

b. Press space bar 2 times

c. Press RETURN 1 time

d. Press your nose 1 time

e. Press letter *m* 10 times

f. Press left arrow 10 times

g. Type PRINT ''GOTCHA'' 1 time

h. Press RETURN 10 times

3. LOW-RESOLUTION GRAPHICS

Now that you are more familiar with the Apple keyboard you can try out some of the graphics features that come with Applesoft BASIC. There are two different systems available, low resolution and high resolution. The low-resolution system, which is the subject of this and subsequent sections, offers a screen with 40 columns and 40 rows.

The columns are numbered from 0 to 39, going from left to right, and the rows are numbered from 0 to 39, going from top to bottom (see Figure 3.1).

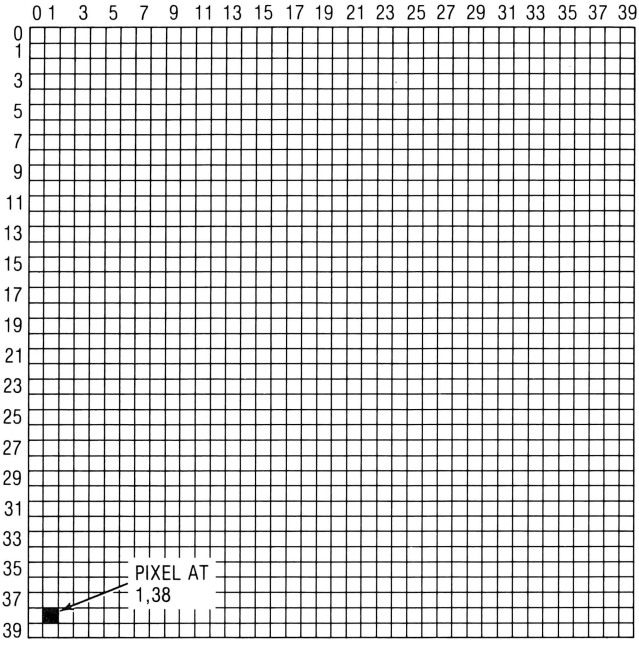


Figure 3.1 Columns and rows in the low-resolution system.

At the intersection of each row and column there lies a pixel, the short name for a picture element. In computer graphics a picture element is the smallest thing that can be displayed on the screen. In the Applesoft low-resolution system each pixel is a small rectangle of solid color, one-half the size of an individual character. Sixteen different colors are available, numbered 0 to 15:

black 0	medium blue 6	green 12
magenta 1	light blue 7	yellow 13
dark blue 2	brown 8	aqua 14
purple 3	orange 9	white 15
dark green 4	grey 10	
grey 5	pink 11	

In order to see one of these colored pixels you must enter the low-resolution graphics mode. When you see the Applesoft prompt, enter

```
GR ⟨RETURN⟩
```

Single keys are shown in angular brackets. The ⟨RETURN⟩ key is like a carriage return key on an electric typewriter, but on a computer it does two things: it enters the lines into memory or performs (executes) whatever you entered after the last ⟨RETURN⟩. You should see the screen clear itself to black. You will also notice that you have only four lines of text window at the bottom. To get back to full-screen text you can press RESET or enter TEXT followed by RETURN.

Before you see the plotted pixel you must set the color to something besides black. If you want orange, for example, enter

```
COLOR=9 ⟨RETURN⟩
```

Now you can plot a pixel. Enter

```
PLOT 0,0 ⟨RETURN⟩
```

You should see a small orange rectangle at the upper left of the screen. If you don't see it, repeat the three steps, GR, COLOR = 9, and PLOT 0,0, with a RETURN after each one.

The other corners of the low-resolution screen are obtained by these commands (see Figure 3.2):

```
PLOT 39,0 (top right)
PLOT 0,39 (bottom left)
PLOT 39,39 (bottom right)
```

Don't forget to press RETURN after each command. You can, in fact, execute more than one com-

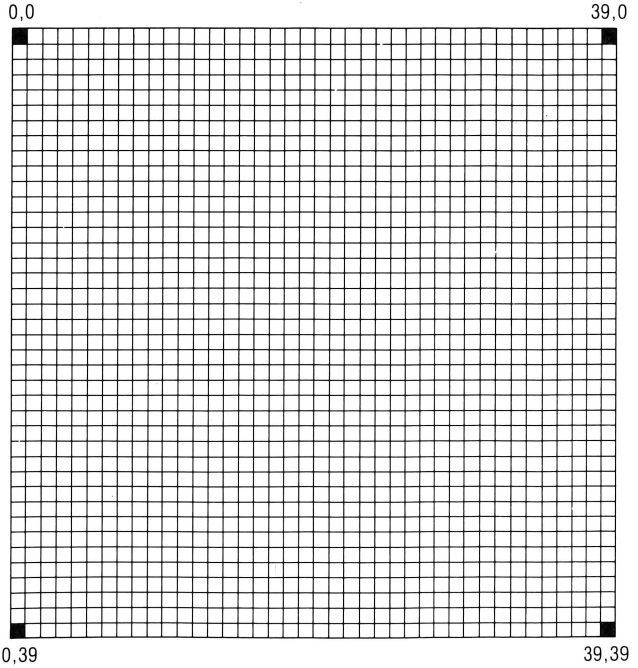


Figure 3.2 The corners of the low-resolution screen.

mand on the same line, if you put a colon (:) between each statement. This line, for example, will put a light blue pixel at the center of the screen:

```
GR:COLOR=7: PLOT 20,20
```

Try plotting pixels of various colors at different locations on the screen, to get used to the Apple low-resolution coordinates, and to see the colors that are available. Adjust your TV set or color monitor so that the colors are correct.

Low-resolution Applesoft BASIC has routines for drawing horizontal and vertical lines. As an example of their use, the following program will draw the boundary of the screen in yellow.

```
10  REM
20  GR : COLOR= 13
30  HLIN 0,39 AT 0
40  HLIN 0,39 AT 39
50  VLIN 0,39 AT 39
60  VLIN 0,39 AT 0
RUN
```

Go ahead and practice with the low-resolution commands. Draw some rectangles of various colors, and then do as many of the exercises as you can.

Exercises

1. Clear the screen to black. Then plot a pink pixel at location 27,15.
2. Draw the LOW RES BOUNDARY with top edge aqua, right edge grey, bottom edge purple, and left edge pink.
3. Outline the LOW RES BOUNDARY with a ribbon of your favorite three colors, each one pixel wide. Is it possible?
4. Select your very favorite color. Then write a program to fill up the entire screen with pixels of this color. *Hint:* Use two FOR . . . NEXT loops, one inside the other, each going from 0 to 39, one with H and the other with V. The idea is to make your program go through all possible pairs of values H,V within the screen. For each pair of values plot the color you selected.

Command	What It Does
GR	Sets low resolution graphics mode with 40 by 40 pixels screen and 4 lines at the bottom for text. The screen is cleared to black and the cursor moves to text window. Full screen 40 by 48 is obtained after GR by POKE −16302,0 or POKE 49234,0
COLOR=	Sets the current color as follows: 0 = black 1 = magenta 2 = dark blue 3 = purple

	4 = dark green 5 = grey 6 = medium blue 7 = light blue 8 = brown 9 = orange 10 = grey 11 = pink 12 = green 13 = yellow 14 = aqua 15 = white
PLOT H,V	Writes a pixel of current color at H-th column and V-th row (H and V between 0 and 39)
HLIN H1,H2 at V	Draws a horizontal line from H1 on the left to H2 on the right at the vertical position V
VLIN V1,V2 at H	Draws a vertical line from V1 at the top to V2 at the bottom at the horizontal position H

4. A COLOR-BAR GENERATOR

Your first project will be to write and understand a short program that will display all 16 colors, each as a vertical bar two pixels wide. Such a pattern is called a color-bar test pattern. They are used by television repair technicians to adjust your TV set properly. You will use an Applesoft BASIC program as your color-bar generator.

First, write this short program to create one vertical bar one pixel wide on the far left of the screen. It should come up yellow.

```
10  REM
20  GR : COLOR= 13
30  FOR V = 0 TO 39
40  PLOT 0,V
50  NEXT
```

If you understand this program completely, skip ahead to the next program, the final color-bar generator. Otherwise, continue reading here, and we will explain all the statements used in the above program, assuming you know nothing about BASIC.

Statement 10 is a REMark statement. It has no effect on your program execution, but it is printed out every time you list your program. It is used for documentation.

Line 20 contains two separate statements (the colon is used to separate more than one statement on a line), the first of which sets the low-resolution graphics mode, and the second of which selects yellow as the color to plot next.

Line 30, FOR V = 0 to 39, instructs the system to perform every instruction between it (the FOR statement) and the nearest NEXT statement, first for the value V = 0, then for the value V = 1, then for V = 2, and so on until V reaches 39. In this case there is only one statement between the FOR and its associated NEXT, line 40,

```
40 PLOT 0,V
```

This statement plots one pixel of the currently selected color at the location zero horizontal, V vertical. Since it is embedded in the FOR . . . NEXT loop, it will be executed exactly 40 times, first with V = 0, then with V = , and so on until V reaches 39.

The color-bar generator is slightly more complicated in that it requires one more loop, one that controls the color plotted. This outer loop will go from C (color) = 0 to C = 15.

Since we have 40 columns available, and 16 colors, we can only afford two columns per color. Black, color number zero, will occupy columns 0 and 1; magenta, color number 1, will occupy columns 2 and 3. In general, therefore, color number C will occupy columns 2C (left-hand pixel) and 2C + 1 (right-hand pixel). If

you count the columns you will see that color number 11 (pink) occupies columns 22 and 23. The last color, number 15 (white), occupies columns 30 and 31. The right 8 columns of the screen remain black for this particular color-bar pattern.

The final program takes this form:

```
10  REM
20  GR
30  FOR C = 0 TO 15
35  COLOR= C
40  FOR V = 0 TO 39
50  PLOT 2 * C,V: PLOT 2 * C - 1,V
60  NEXT : NEXT
```

Go ahead and RUN it. You will see the color bars on your TV screen, unless you have made an error somewhere along the line. If you do get all fifteen visible bars, the only remaining task is to adjust your TV set.

On most color TV sets there are four controls that determine color values: *chroma* or *hue, saturation* or *tint, brightness,* and *contrast.* First adjust the hue or chroma control so that all colors are in the color category indicated by their name. For example, pink should have a red tinge; light or dark blue should have a blue tinge, not green or orange.

Next adjust the saturation or tint control so that each bar has a comfortable amount of color, not so saturated that it spills out into other colors, but not so dilute that it seems pale.

Finally, adjust the brightness and contrast controls for best viewing in the available light around the TV screen.

Exercises

1. Draw a grey line, one pixel wide, across the screen at the top.
2. Draw a ribbon across the center of the screen by making row 17 green, row 18 orange, and row 19 aqua.
3. Modify the example program ***YELLOW VERTICAL BAR*** so that the bar is purple.
4. Modify the example program ***COLOR-BARS*** so that the bars run horizontally instead of vertically.
5. Shorten ***COLORBARS*** by using VLIN.
6. Pick a pair of colors that you like to see together. Then try a screenful of alternating horizontal bars of those two colors, each bar 3 pixels wide.
7. Automate your program in exercise 6 so that the user can input the color pair, the width of the bar, and the orientation (horizontal or vertical).

8. Write a program called ***COLOR-SQUARES*** that puts each of the 16 colors on the screen, in a square one pixel wide.

5. FLAG DESIGNS

There is a simple way to fill up the low-resolution screen with flag designs that are quite impressive to look at. For ease of programming we will make our flags square, with 40 pixels on a side. We will start with a flag that is all one color, color number one, magenta.

```
10  REM
20  GR
30  FOR H = 0 TO 39
40  FOR V = 0 TO 39
50  COLOR= 1
60  IF H = V - 1 OR H = V + 1 OR H =
    38 - V OR H = 40 - V THEN  COLOR= 9
70  IF H = V OR H = 39 - V THEN  COLOR= 5
80  PLOT H,V
90  NEXT : NEXT
```

Notice that the flag, even though it ends up a uniform color, is drawn in real time exactly as your program commands. In line 30 the outer FOR . . . NEXT loop is controlling the horizontal coordinate, which moves from left to right across the flag. At each value of H the inner loop, controlled by the vertical coordinate V, goes through every vertical position from 0 to 39, as specified in line 40. Whatever block of statements we end up putting between line 40 and line 110 will be executed once for each of the possible positions H,V in our flag.

If you want to see a screen full of one particular color, change line 50 to COLOR = (your choice) and RUN the program again.

The idea is to add statements one at a time and RUN the program after each new statement is added, so that we can see its effect.

For example, the top edge of the screen is defined by H = any value between 0 and 39, V = 1. In other words, if the vertical coordinate of a pixel is 0, you know it is from the top row of the screen. If you add this statement:

```
55 IF V=0 THEN COLOR=7
```

and RUN the resulting program, you should see the top edge in light blue.

If you want the entire screen boundary in light blue you can enter a more complicated version of the above statement:

```
55 IF V=0 OR V=39 OR H=0 OR H=39 THEN COLOR=7
```

Go ahead and try this. Keep working on it until you get a magenta flag with a light-blue border.

The (imaginary) line running from the top left corner to the bottom right corner is called the *main diagonal* of the screen (see Figure 5.1). It is defined

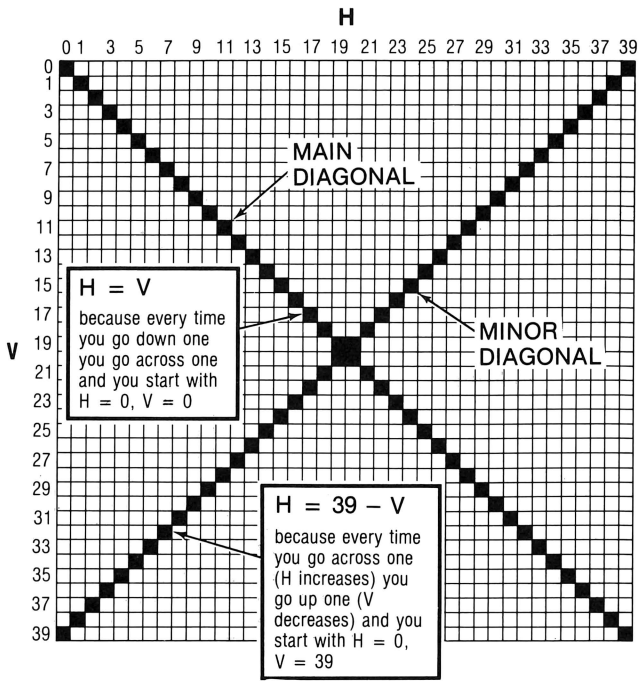

H
0 1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39
V
0 1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39
MAIN DIAGONAL
H = V
because every time you go down one you go across one and you start with H = 0, V = 0
MINOR DIAGONAL
H = 39 – V
because every time you go across one (H increases) you go up one (V decreases) and you start with H = 0, V = 39

Figure 5.1 Main diagonal and minor diagonal.

by the requirement that H = V, or, you go right one every time you go down one. If you want to see it in your favorite color, add this statement:

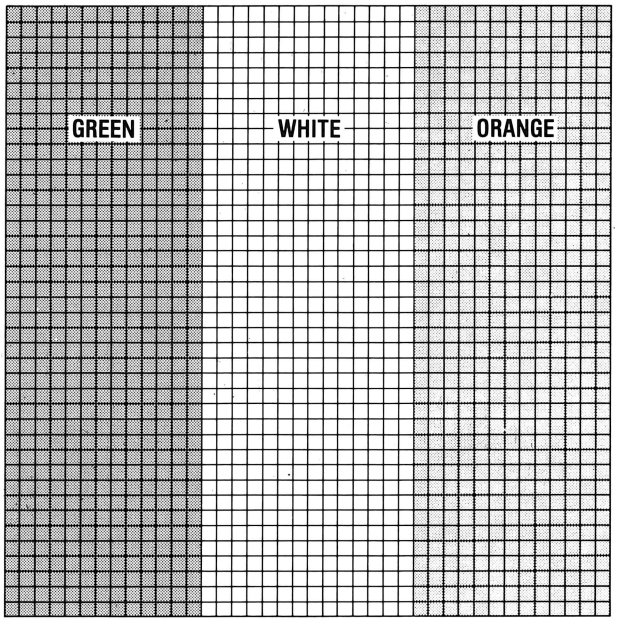
```
60 IF H=V THEN COLOR=(enter the number of
   your color)
```

The other diagonal (from lower left to upper right) is called the *minor diagonal*. Try to guess what the relation between H and V is.

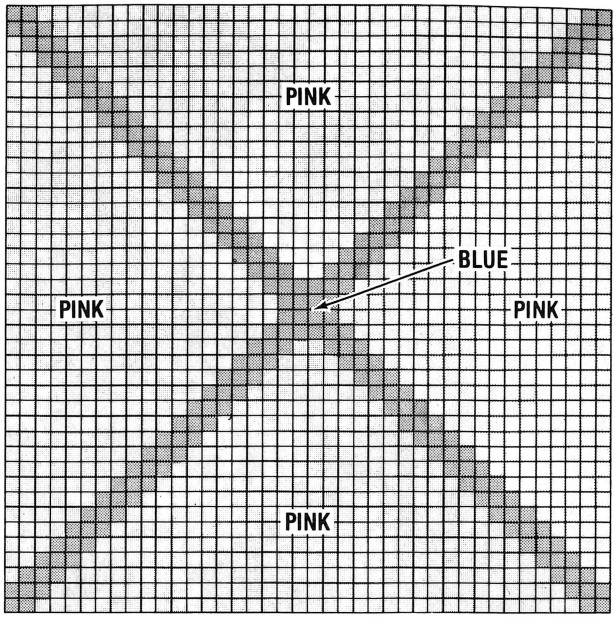
Exercises

Try creating these flag designs.

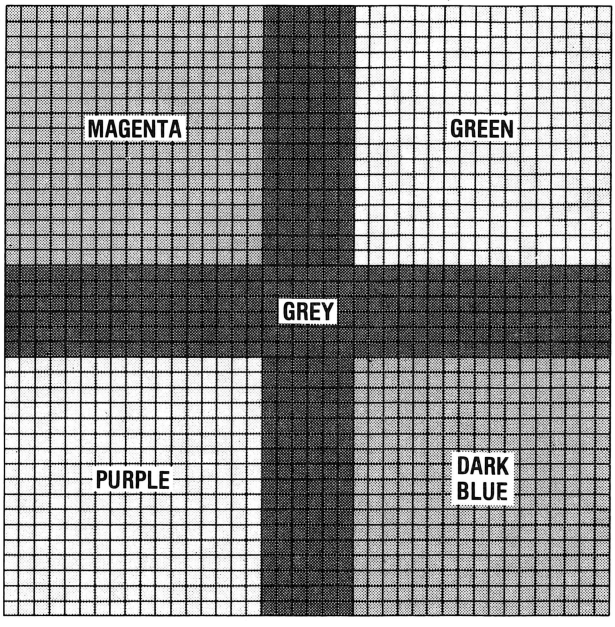
1.



2.



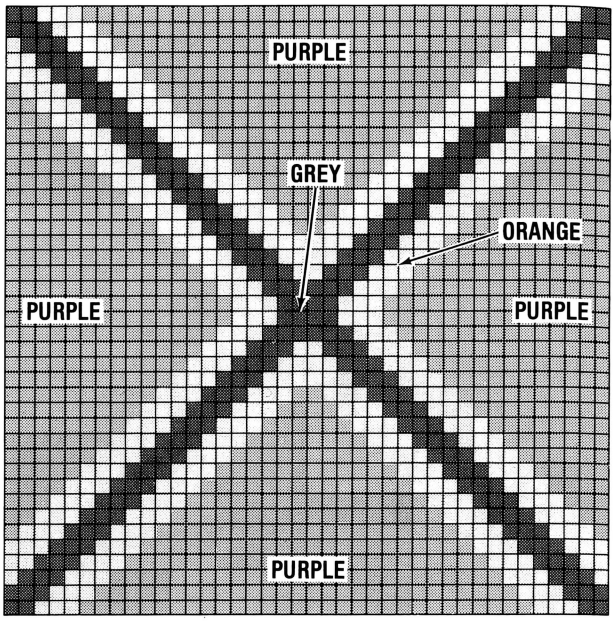
3.



Answer:

```
10  REM      ***FLAG EXERCISE 3***
20  GR
30  FOR H = 0 TO 39
40  FOR V = 0 TO 39
45  COLOR= 10: REM  GRAY
50  IF H = 15 AND V = 15 THEN  COLOR= 1:
    REM MAGENTA
60  IF H = 25 AND V = 15 THEN  COLOR= 12:
    REM GREEN
70  IF H = 15 AND V = 25 THEN  COLOR= 3:
    REM PURPLE
75  IF H = 25 AND V = 25 THEN  COLOR= 2:
    REM DARK BLUE
80  PLOT H,V
90  NEXT : NEXT
```

4.

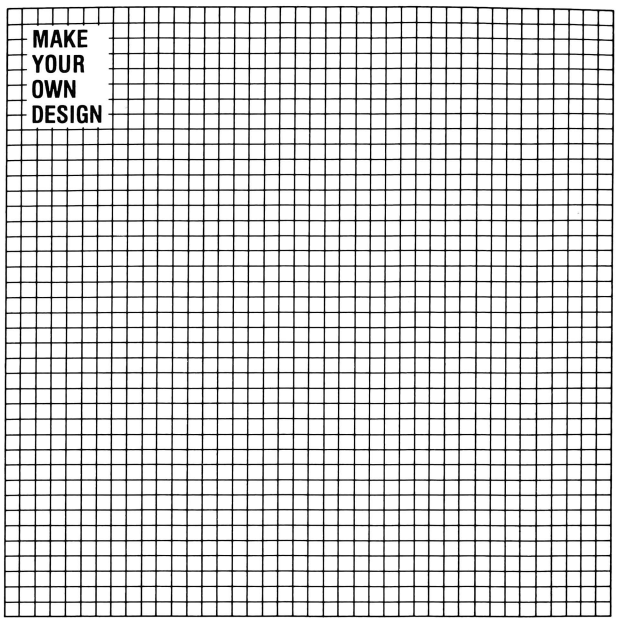


5.

Answer:

```
10  REM    ***FLAG EXERCISE 5***
20  GR
30  COLOR= 7
40  FOR V = 0 TO 39
50  HLIN 0,39 AT V
60  NEXT V
70 :
80  COLOR= 6
90  FOR V = 10 TO 30
100  HLIN 10,30 AT V
110  NEXT V
120 :
130  COLOR= 2: FOR V - 15 TO 25: HLIN 15,25
    AT V: NEXT V: VTAB 21: CALL  -
     958: VTAB 23:
    HTAB 6: FLASH : PRINT - WOW!!  , ,
    A BIG BLUE
    BLOCK!! ": NORMAL : GOTO 130
```

6.



6. HOW TO MAKE A KALEIDOSCOPE

There is a simple toy that has been around for quite some time. It is called a kaleidoscope, and it looks like a cardboard tube that you hold to your eye. Inside are colorful rocks of various sizes and shapes, and a mirror and lens system that reproduces the same segment 8 or 16 times around the circle. This produces a pattern of high radial symmetry, which is enjoyable to look at. As you twist the cardboard tube the rocks move and the pattern changes.

We will show you how to make your own kaleidoscope using Applesoft BASIC. The main idea is to use pixels as rocks, painting each one a random color, and placing it at a random location on the screen. This can be accomplished by the following lines of code:

```
10  REM
20  GR
30  COLOR=  INT (16 *  RND (1))
40 H =  INT (40 *  RND (1))
50 V =  INT (40 *  RND (1))
60  PLOT H,V
70  GOTO 30
```

Type this in and run it, and watch the screen fill up with pixels. Notice that the pattern looks quite random at this point.

You may be wondering how lines 30, 40, and 50 work, since they involve two library functions, INT (the integer function) and RND (the random number function).

Here's how INT works. Inside the parentheses that go with the INT function will be an expression that evaluates to a floating point number, like 4.234527. The value returned by INT is the largest integer not greater than the number inside the parentheses. Thus INT (3.25) is 3.

The RND function when applied to the argument 1 returns a random floating point number somewhere between 0 (which is included) and 1 (which is not included). Every time you use the function you get a new number.

Now let us look at line 30 in this light. When we multiiply the random number (it's never exactly 1) by 16 we get numbers that range from just above zero to just below 16. Their integer values therefore will fall on one of the numbers 0, 1, 2, . . . 15. Thus each time statement 30 is executed a new color (including black or white) is enabled.

In a similar way lines 40 and 50 set H and V to random integers between 0 and 39.

In line 50 we plot the pixel at H,V and then go back to repeat the entire process over again.

Incidentally, the easiest way to stop these infinite loop test programs is to press RESET. This also gets you back into TEXT mode so you can list your program.

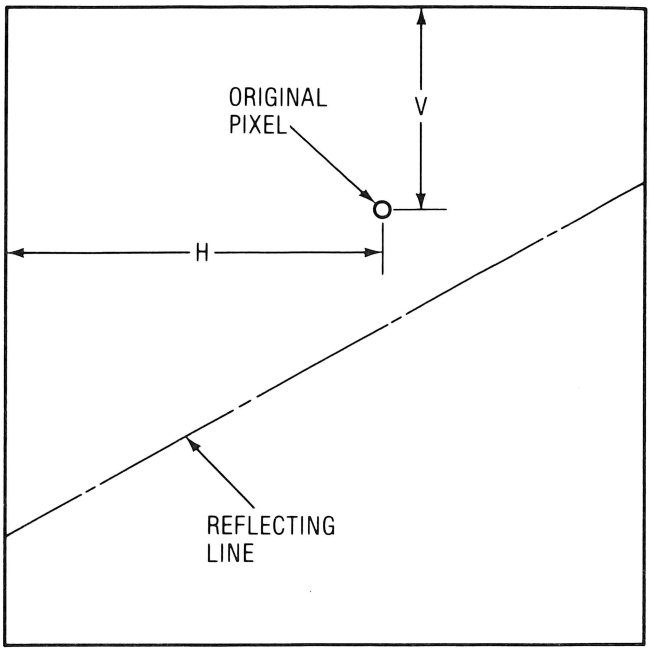


Figure 6.1 Finding the reflecting line.

In simulating a kaleidoscope on the Apple we will use reflecting lines instead of mirrors. These have a simpler geometry. You can find the reflection in a few steps (see Figure 6.1).

Step 1.

Locate the reflecting line and the pixel to be reflected. Draw it as a small circle. Draw in dimension lines from the pixel to the origin and label them H and V.

Step 2.

Draw a line from the original pixel perpendicular to the reflecting line and crossing it. Go beyond the reflecting line an equal distance and place a small star. This is the reflected pixel. Draw in dimension lines from the reflection to the origin, and label these H2 and V2 (see Figure 6.2).

Step 3.

Derive expressions for the coordinates H2,V2 of the reflection in terms of the coordinates H,V or the original pixel.

The formula for the reflected coordinates H1,H2 in terms of the original coordinates can be simple or complex, depending on where the reflecting line is placed. If the reflecting line is placed horizontally in the center of the screen the transformation is relatively simple (see Figure 6.3).

To see this, notice first that the horizontal coordinate of the reflected image will be the same as the horizonal coordinate of the original image. Then notice that the distance from the reflected image to the bottom of the screen is V, because of where the reflection

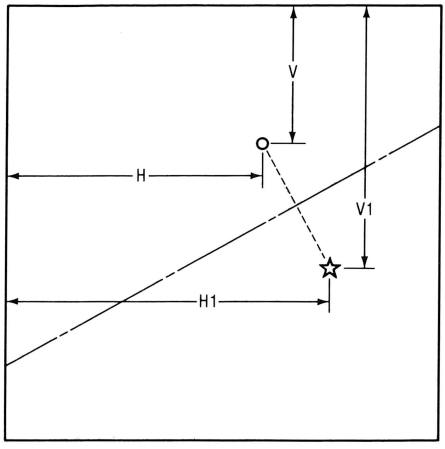


Figure 6.2 Finding the reflected pixel.

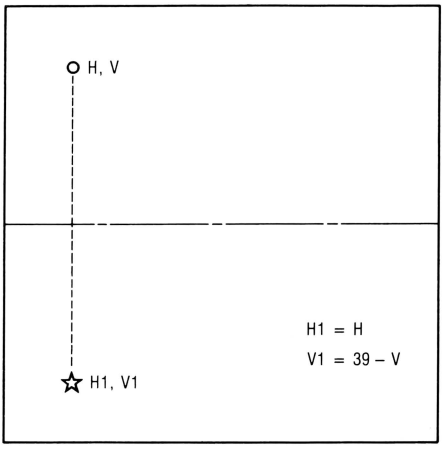


Figure 6.3 Transformation from a horizontal reflecting line.

is. Thus the distance from the top of the screen to the reflected image is the screen height minus V.

If the reflecting line is placed along the vertical center line a similar thing happens, with horizontal and vertical reversed (see Figure 6.4).

So far we have two reflections of the original pixel (see Figure 6.5).

You can enter these reflections by adding the lines

```
70 PLOT 39-H,V
80 PLOT H,39-V
```

If you type RUN after line 80 you should get three pixels on the screen. Every time you RUN it they will appear in different locations. Do it several times to make sure the reflections are right. To get the fourth pixel we want to reflect one of the reflections across the other reflecting line. This will do it:

```
90 PLOT 39-H,39-V
```

Now test your program with these two reflecting lines in place. Simply enter

```
100 GOTO 30
RUN
```

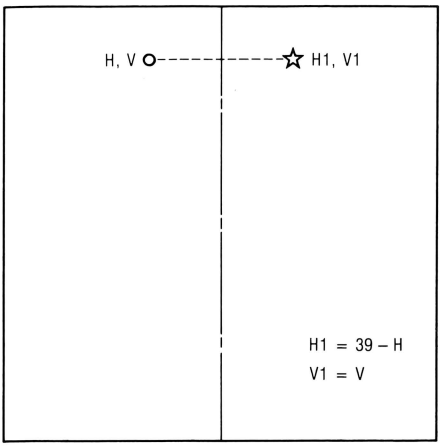


Figure 6.4 Transformation from a vertical reflecting line.

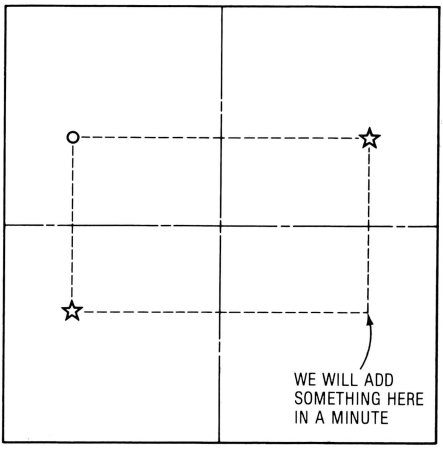


Figure 6.5 Two reflections for the original pixel.

Let the screen fill up, and watch it for a while. Show it to a friend.

The final step is to reflect each of the four pixels across the main diagonal (see Figure 6.6).

You can imagine the screen as a flat piece of cardboard, which you are holding at the top left and bottom right corners. To reflect about the main diagonal you

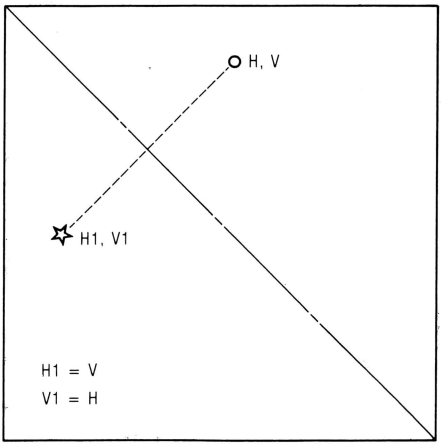


Figure 6.6 Four pixels reflected across the main diagonal.

flip the cardboard over. What was the left vertical edge becomes the top horizontal edge, and vice versa.

Therefore we can repeat our four plotting lines, after interchanging horizontal and vertical coordinates:

```
100 PLOT V,H
110 PLOT V,39-H
120 PLOT 39-V,H
130 PLOT 39-V,39-H
```

Before you add the GOTO 30, test it with a few RUNs. Each time you run it you should get 8 pixels, all the same color (if the color selected is black you won't see anything), arranged in an eightfold symmetrical pattern.

Then just add

```
140 GOTO 30
RUN
```

and sit back to watch it. After the screen fills up call some friends in to see what you have done (Plate II).

Exercises

1. Pick two colors that you like to see together. Modify line 30 so that it sets the color to one of these two colors. Then run the kaleidoscope again. Wait for it to fill up. *Hint:* The expression

   ```
   30 X=INT(RND(1)+.5)
   ```

   will return either 0 or 1, depending if the number returned by RND(1) was less than .5 or more than .5. You can then use this information to select the colors (say 3 and 12):

   ```
   32 IF X=0 THEN COLOR=2
   34 IF X=1 THEN COLOR=15
   ```

   (If you don't have a color monitor, the best contrast should be:

   2 dark blue 0 black
   13 yellow 15 white
   5 grey

   because many of the other colors don't plot a ''filled-in'' pixel in monochrome.

2. Restructure the kaleidoscope program using a subroutine that plots the four points. Then call this subroutine twice for each set of H, V and COLOR values.

Answer:

```
10  REM  ***KALEIDOSCOPE EXERCISE 2***
20  GR
30 X = INT ( RND (1) - .5)
32  IF X = 0 THEN  COLOR= 2
34  IF X = 1 THEN  COLOR= 15
40 H = INT (40 *  RND (1))
50 V = INT (40 *  RND (1))
```

```
70  GOSUB 100:TEMP = H:H = V:V = TEMP:
    GOSUB 100: GOTO 30
100  PLOT V,H
110  PLOT V,39 - H
120  PLOT 39 - V,H
130  PLOT 39 - V,39 - H
140  RETURN
```

3. (Advanced students) Make a kaleidoscope with 16 sections by reflecting the eight pixels across a reflecting line that passes through the screen center at an angle halfway between horizontal and the main diagonal (inclined 22.5 degrees). You may need to ask a math teacher to help you with this one.

Answer:

```
100  REM   *** KALEIDOSCOPE EXERCISE  3 ***
110  REM
120  GR
130  COLOR=  INT ( RND (1) * 14) - 1
140  REM
150  REM ----- FIND H1 & V1
160  REM
170 H1 =  RND (1) * 20
180 V1 = 20 -  RND (1) * ((20 -
      H1) * 8.07 / 20)
190  REM
200  REM ----- FIND H2 & V2
210  REM
220 RAD =  SQR ((20 - V1) - 2 - (20 - H1)
     - 2): REM  FIND COMMON RADIUS TO BOTH
     H1,V1 & H2,V2
230 A =  SIN ((20 - V1) / RAD): REM FIND
     COMMON ANGLE.
240 B = A - (22.5 * 3.14 / 180) * 2
250 H2 =  INT (20 -  COS (B) * RAD)
260 V2 =  INT (20 -  SIN (B) * RAD)
270  REM
280  REM ----- PLOT
290  REM
300  GOSUB 360: REM  PLOT FIRST (8)
     SECTIONS.
310 H = H1:H1 = V1:V1 = H:::H = H2:H2 =
     V2:V2 = H: REM SWITCH HORZ. -->
     VERT. VALUES.
320  GOSUB 360: REM PLOT SECOND (8)
     SECTIONS.
330  GOTO 130: REM GO DO IT AGAIN!!
340  REM
350  REM
360  PLOT H1,V1
370  PLOT H1,39 - V1
380  PLOT 39 - H1,V1
390  PLOT 39 - H1,39 - V1
400  REM
410  PLOT H2,V2
420  PLOT H2,39 - V2
430  PLOT 39 - H2,V2
440  PLOT 39 - H2,39 - V2
450  RETURN
```

4. (Advanced students) Instead of pixels, use HLIN and VLIN to generate random-length (uniformly colored) horizontal and vertical bars. For each such bar generate the seven other reflections of it. This kaleidoscope has pizzazz.

5. (Graduate students) Ask the user how many reflections are wanted. Then generate a kaleidoscope with that number of reflecting lines (placed at equal angles and passing through the center of the screen). This is an excellent project because of the approximations that are involved. To continue the project, switch to high resolution and increase the number of mirrors.

7. LOW-RESOLUTION LINES

As you have seen, there is no problem drawing straight lines on the low-resolution screen, as long as they are horizontal or vertical. You have the choice of using the Applesoft routines HLIN or VLIN, or programming the loops yourself.

When it comes to drawing straight lines that are not horizontal or vertical, however, you will not be able to get smooth-looking lines, because the pixels are so large. This is probably why Applesoft contains no line-drawing routines (except HLIN + VLIN) for low resolution.

Suppose you want to draw a straight line that starts at H1,V1 and ends at H2,V2 (see Figure 7.1).

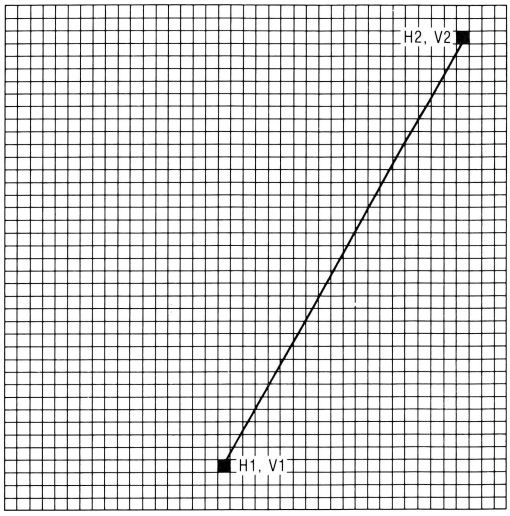


Figure 7.1 The ideal line between two points.

The line that we are approximating is called the *ideal line* between the two points. We can think of it as being drawn on an ideal screen placed under the low-resolution screen. On the ideal screen points can be drawn with nine-digit accuracy (the accuracy of Applesoft floating-point arithmetic, which we use to simulate the ideal screen).

The ideal line starts at the center of the pixel at H1,V1 and ends at the center of the pixel at H2,V2. The start and end coordinates of the ideal line will be the same relative to the ideal screen.

The rule we plan to use is simple: If the ideal line cuts across a pixel, plot that pixel. Figure 7.2 shows another example of the rule.

Your program begins by getting the start and end coordinates from the user:

```
10  REM
15  GR : COLOR= 15
20  INPUT "H1= ";H1
30  INPUT "V1= ";V1
40  INPUT "H2= ";H2
50  INPUT "V2= ";V2
60  PRINT "THANKS"
```

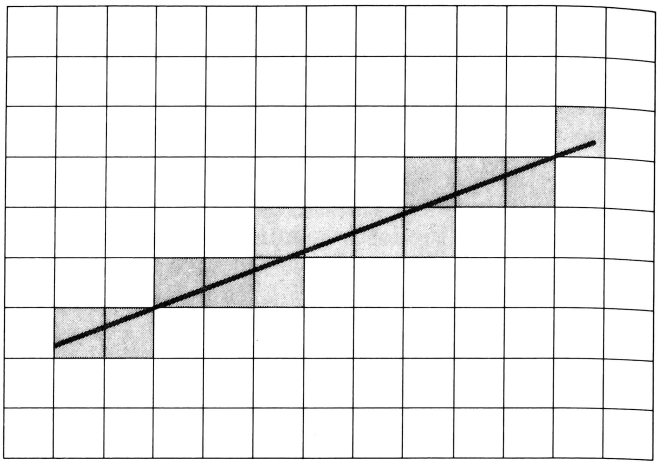


Figure 7.2 If the ideal line cuts across a pixel, plot that pixel.

Go ahead and test it so far. Remember that H1 is the horizontal coordinate, and V1 is the vertical coordinate, of the start of the line. H2 and V2 represent the end of the line. See what happens if you put in coordinates that are less than zero or more than 39. What happens if you put in decimal coordinates, like 3.24?

The ideal screen we will use in the program will be simulated by using floating-point arithmetic. We must draw the ideal line on this nine-digit precision ''screen'' by storing some sort of an equation that represents the line.

One easy way to represent a straight line is to calculate its *slope*, *M*. The slope of a line measures the angle it makes with a horizontally placed line. We define the slope as the vertical distance traveled, V2 – V1, divided by the horizontal distance traveled, H2 – H1.

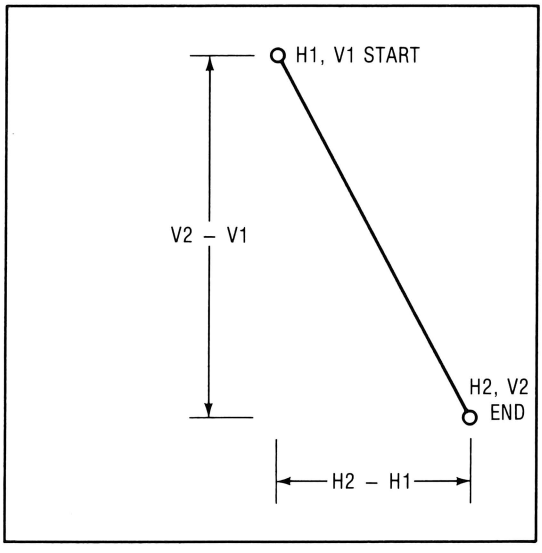


Figure 7.3 Calculating slope.

For now add the line as shown in Figure 7.3 and test it.

```
70  M = (V2 - V1) / (H2 - H1): PRINT M
```

Now we need to talk about what this number, the slope, means. For one thing, if it is positive that means the line is sloping downward. In the coordinate system that you may have learned in school a positive slope means the line is going upward. In microcomputer graphics, however, the vertical direction is positive doing down. The best way for you to reconcile this difference is to think only in Applesoft.

A slope of 1 produces a line that heads down at a 45-degree angle. Each succeeding increase in slope produces a smaller change in angle (see Figure 7.4). If you attempt to input a vertical line, where H1 = H2, no slope can be calculated.

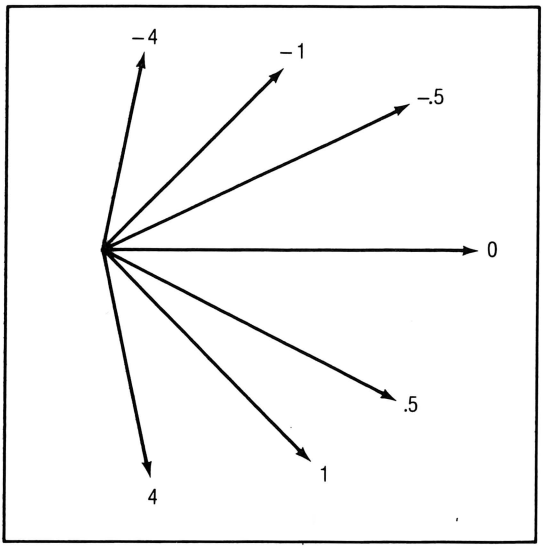


Figure 7.4 Increasing slope produces a smaller change in angle.

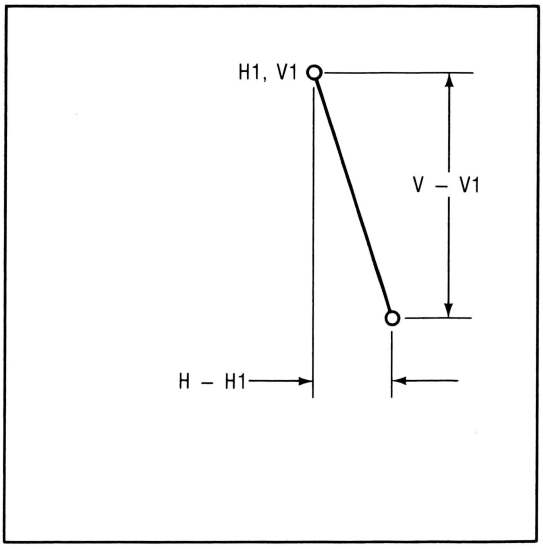


Figure 7.5 The line never changes direction, therefore the slope is the same.

Now let H,V be a point that moves along the ideal line. We know that the slope as computed from H1,V1 to H,V must be the same as the slope computed over the entire line, because the line never changes direction, as shown in Figure 7.5. In other words,

```
(V-V1)/(H-H1)=M
```

for all points H,V along the ideal line. Thus if we know H we can solve for V. The required expression is:

```
V=M*(H-H1)+V1
```

This equation is very important. It allows us to plot the vertical coordinate of a point on the ideal line if we are given the horizontal coordinate.

We plan to move H between H1 and H2 in steps of .05, or 20 samples across the width of each pixel. For each value of H we compute V, and then plot the pixel on which the point H,V lands. Many times we will be plotting the same pixel over and over again, and the calculations are performed much more often than would be necessary using another approach. But the program works. More efficient solutions become quite complex, as the exercises demonstrate.

```
80  FOR H = H1 TO H2 STEP .05
90  V = M * (H - H1) - V1
100  PLOT  INT (H - .5), INT (V - .5)
110  NEXT
120  GOTO 20
```

In line 100 we have added .5 to the horizontal and vertical ideal coordinates because H and V will be starting in the corner of a pixel. When H exceeds H1 + .5 it enters the next pixel. We want the INTeger value to be changing upward at that point.

Go ahead and debug the program. Try it on several lines with different slopes.

Exercises

1. See what happens when you try to plot the line between 25,39 and 26,0. You will discover some pixels that didn't get displayed. Study the problem and fix it. This can be done by modifying line 80; no other changes are required.

Answer:
```
80 FOR H=H1 TO H2 STEP .02
```

2. Modify LOW RES LINES so that it plots a random color each time it goes to line 100.

Answer:
```
95 COLOR=INT(RND(1)*15)
```

3. Modify LOW RES LINES so that it works from random inputs, but draws each line in the same random color. You should be able to let the program run all day.

Answer:
```
20 H1=RND(1)*39
30 V1=RND(1)*39
40 H2=RND(1)*39
50 V2=RND(1)*39
60 COLOR=INT(RND(1)*15)
```

4. Rewrite LOW RES LINES so that it goes much faster. There are several ways you can do this, and each way presents a challenge. This exercise is ideal for a programming contest. Whoever can plot the same set of selected lines fastest wins. The set of selected lines must not favor any particular method.

Answer:

```
10  REM     *** EXERCISE  4  ***
20  DEF  FN A(C) =  RND (1) * C
30  H1 =  FN A(39):V1 =  FN A(39):H2 =  FN
    A(39):V2 =  FN A(39): GR : COLOR=
    FN A(15):M = (V2 - V1) / (H2 - H1):
    FOR H = H1 TO H2 STEP .5:V = M * (H -
     H1)
    - V1: PLOT  INT (H - .5), INT (V -
     .5): NEXT : GOTO 30
```

5. Modify LOW RES LINES so that it works in high resolution. If you are not yet familiar with high-resolution graphics, read the next section and then come back to this exercise.

Answer:

```
10  REM  *** EXERCISE  5  ***
11  REM
20  INPUT "H1= ";H1
30  INPUT "V1= ";V1
40  INPUT "H2= ";H2
50  INPUT "V2= ";V2
60  PRINT "THANKS"
65  HGR : HCOLOR= 3
70  M = (V2 - V1) / (H2 - H1): PRINT M
80  FOR H = H1 TO H2
90 V = M * (H - H1) - V1
100  HPLOT H,V
110  NEXT H
120  GOTO 20
```

6. (For math majors) You can also generate a straight line parametrically, by making both H and V functions of another variable, which we shall call *P*, for *parameter*. In this case your program asks the user for the starting point (H1,V1), the *bearing* (in degrees counterclockwise from due east), and the *range* (in pixels). Use the SIN and COS functions in Applesoft BASIC to derive the coordinates of a point along the line.

8. HIGH-RESOLUTION GRAPHICS

The Apple // also has a high-resolution system, with 280 horizontal points (from 0 to 279) and 160 (from 0 to 159) vertical points. Each point can be one of six (eight different) colors:

0	black1	4	black2
1	green	5	pink (depends on TV)
2	blue	6	orange (depends on TV)
3	white1	7	white2

There are actually two separate high-resolution screens, the regular one, HGR, and a second one, HGR2. The second screen, HGR2, does not have text lines at the bottom, so its vertical coordinate can go to 191, for a total of 192 vertical lines.

The command HGR clears the high-resolution screen 1 to all black, leaves four lines at the bottom for program-generated printout, and waits to write on screen 1.

The command HGR2 clears high-resolution screen 2 to black and sets screen 2 as the screen that will be written on next.

In high-resolution graphics the color is set by an HCOLOR= command (with values 0 to 7) and plotting is done with the HPLOT command. The following program, for example, will draw a white boundary on high-resolution screen 2.

```
10 REM ***HIGH*RES*BOUNDARY***
20 HGR2: HCOLOR=3
30 FOR H=0 to 279:
40 HPLOT H,0: HPLOT H,191
50 NEXT
60 FOR V=0 TO 191
70 HPLOT 0,V: HPLOT 279,V
80 NEXT
RUN
```

Let us review how this program works. Line 20 clears HGR2 to black, prepares to write on it, and selects the color white1. Lines 30 to 50 comprise a FOR . . . NEXT loop which steps the horizontal coordinate H from zero (far left) to 279 (far right) in steps of one. For each value of H we plot (in line 70) first the topmost point at that horizontal coordinate, then the bottom-most point at the same horizontal coordinate. Lines 60 to 80 draw the left and right vertical boundaries using the same technique.

Watching this program run will give you an idea of the plotting speed of Applesoft in the high-resolution mode using HPLOT within a FOR . . . NEXT loop.

This same task, drawing the boundary of high-resolution screen 2, can be accomplished using a more powerful variation of the HPLOT command. Try it out first; then we will discuss it.

```
10 REM ***HPLOT EXAMPLE*****
20 HGR2: HCOLOR=3
30 HPLOT 0,0: HPLOT TO 0,191
40 HPLOT TO 279,191: HPLOT TO 279,0
50 HPLOT TO 0,0
```

Notice how much faster the boundary is drawn. This is because the plotting is done at the level of machine, or assembly, language. Once it knows what to do it jumps out of BASIC and follows machine code until the line is drawn; then it jumps back into following your BASIC program.

The HPLOT TO command draws a straight line, in whatever COLOR is currently set, from the last point plotted to the coordinates following TO. You must precede HPLOT TO H,V with a previously plotted HPLOT H,V point or HPLOT TO H,V will *not* be plotted and you will get a blank screen. This will be useful, in FOR-NEXT loops, when plotting fast circles.

The most general form of the HPLOT command indicates both the starting and the ending points between which a straight line is to be drawn. Add line 60 to the above program and run it; it will add a blue minor diagonal to your white boundary.

```
60 HCOLOR=2: HPLOT 0,191 TO 279,0
```

To review HPLOT: You can plot a single point, draw a line from the last point to the specified point, or draw a straight line between two specified points.

Exercises

1. Use HPLOT to draw a grey boundary on high-resolution screen 1. Remember the vertical limit is 159 for screen 1.

2. Draw a series of random lines in random colors (exclude black). *Hint:* to get a random integer between 0 and 279 use the expression INT(280*RND(1)). Do a similar thing to get a random vertical coordinate, and a random color 0 to 7. Use HCOLOR=(random) and HPLOT to (random),(random) in a loop that keeps repeating (see Plate III).

Answer:

```
10  REM    ***EXERCISE 2***
20  REM
30  HGR : HCOLOR= 1
40  FOR H = 124 TO 208
50  HPLOT H,78 TO H,136
60  NEXT
```

3. Consider the rectangular area of the screen, where rows 78 to 136 intersect columns 124 to 208. Draw this area in green against an all-black background.

Answer:

```
10  REM   ***EXERCISE 3***
20  REM
30  REM    !!!  ON A B/W SCREEN THERE WILL
    ONLY BE B/W !!!
40  REM
50  REM
60  HGR
70  FOR H = 0 TO 279
80  C = (C - 1) * (C - 7): HCOLOR= C
90  HPLOT H,0 TO H,159
100  NEXT
```

4. Draw vertical lines one pixel wide, using all colors 0 to 7, from left to right. Repeat this rainbow as many times as it takes to fill the screen (about 36 times).

Answer:

```
10  REM   ***EXERCISE 4***
20  REM
30  REM    !!!  ON A B/W SCREEN THERE WILL
    ONLY BE B/W !!!
40  REM
50  REM
60  HGR
70  FOR I = 0 TO 9E9
80 C = (C - 1) * (C - 7): HCOLOR= C
90  HPLOT 0 - I,0 - I TO 0 - I,159 - I
100  HPLOT  TO 279 - I,159 - I
110  HPLOT  TO 279 - I,0 - I
120  HPLOT  TO 0 - I,0 - I
130  NEXT
```

5. Do the same sort of thing as in exercise 3 above, but use rectangles of each color, and fit the next rectangle just inside the last rectangle. Keep going toward the center until you run out of space.

Answer:

```
10  REM   ***EXERCISE 5***
20  REM
30  REM    !!!  ON A B/W SCREEN THERE WILL
    ONLY BE B/W !!!
40  REM
50  REM
60  INPUT "NUMBER OF PIXELS WIDE? :";NP
70  HGR
80  FOR I = 0 TO 9E9
90  IF I = 159 THEN I = I - 159: REM THIS
    EQUATION ALLOWS PLOTTING TO
    CONTINUOUSLY.
100  C1 = C1 - 1: ON C1 = NP
    GOTO 120:C1 = 0: REM  COUNTS NUMBER OF
    LINES
    THAT ARE THE SAME COLOR.
110  C = (C - 1) * (C - 7): HCOLOR= C:
    REM INCREMENTS COLOR CONTINUOUSLY THRU
    - -> 7
120  HPLOT 0 - I,0 - I TO 0 - I,159 - I
```

```
130  HPLOT  TO 279 - I,159 - I
140  HPLOT  TO 279 - I,0 - I
150  HPLOT  TO 0 - I,0 - I
160  NEXT
```

6. Repeat exercise 4 again with this new feature: Let the user input how many pixels wide each color should be. This same feature can be added to exercise 3.

7. Draw a flag consisting of a white border and white major and minor diagonals, using 0 to 159 in each direction on a black background. Write two programs. Number 1 should use HPLOT and be as fast as possible, and number 2 should use a pair of nested FOR . . . NEXT loops and IF statements, like the low-resolution flag design programs. Use a stopwatch to compare the run times of 1 and 2.

Answer:

```
10  REM  ***EXERCISE 7***
20  REM
30  REM
40  HGR : HCOLOR= 3
50  REM
60  REM ----- PLOT BORDER
70  REM
80  FOR I = 0 TO 9
90  HPLOT 0 - I,0 - I TO 0 - I,159 - I
100  HPLOT  TO 279 - I,159 - I
110  HPLOT  TO 279 - I,0 - I
120  HPLOT  TO 0 - I,0 - I
130  NEXT
140  REM
150  REM ----- PLOT DIAGONALS
160  REM
170  FOR J = 0 TO 4
180  HPLOT 5 - J,0 - J TO 279 - J,154 - J:
     REM MAJOR DIAGONAL.
190  REM
200  HPLOT 0 - J,154 - J TO 274 - J,0 - J:
     REM MINOR DIAGONAL.
210  NEXT J
```

8. (Advanced students) Draw the outlines of diamonds of random size (test to make sure they fit on the screen) and put them at random locations in random colors. Make your program as fast as possible, and draw only complete diamonds. If you really want to get fancy you can tilt each diamond a small but random amount, which is barely perceptible. This gives the drawing some snap.

9. HOW TO DRAW A CIRCLE

Sooner or later you will want to create graphic designs that include circles. Unless you have a graphics package on your system you will have to make the circles yourself, using the geometric definition of a circle.

A circle of radius R with center at H,V is the set of points that are a distance of R away from the center. Because of this definition, circles are easy to draw with a compass, a mechanical device with a sharp pin on one end and a pencil on the other, and with a fixed but adjustable distance, the radius, between the pin and the pencil. The compass has a handle at the top which is rotated through 360 degrees while gentle pressure is applied. The pencil point traces out the circle as it is spun around (see Figure 9.1).

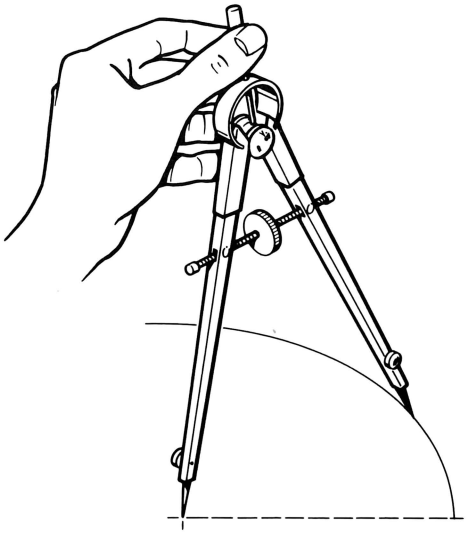


Figure 9.1 Drawing a circle with a compass.

We assume that you will start the compass pencil point to the far right of the circle center, and that you will twist the compass handle in a counterclockwise direction. This corresponds to the way in which engineers measure angle. When the pencil is to the right, the line from center to pencil points to zero degrees. In math terms, therefore, zero degrees points due east. North, or straight up, is 90 degrees, west is 180 degrees, south is 270, and east is 360 (or any multiple of 360).

There is another way we measure angle that you must know about, because the functions that you will be using in Applesoft BASIC use it. In this system angle is measured in *radians*.

In the radian system, halfway around the circle is pi radians, that is, 3.14159. The full 360 degrees contains twice pi radians.

Because we are moving the compass by spinning it across a certain angle, and because Applesoft BASIC must know the horizontal and vertical distances from the screen origin to the pencil point, we must convert angle to distance. This requires a little trigonometry.

All you need to know is this: The SINe of an angle is the opposite over the hypotenuse; the COSine of an angle is the adjacent over the hypotenuse (see Figure 9.2).

Let *A* be the angle in radians that you have twisted the compass handle from its initial bearing of zero radians. We need to compute *X* and *Y*, the horizontal and vertical distances from the screen origin to the pencil point. The geometry of the situation is shown in Figure 9.3.

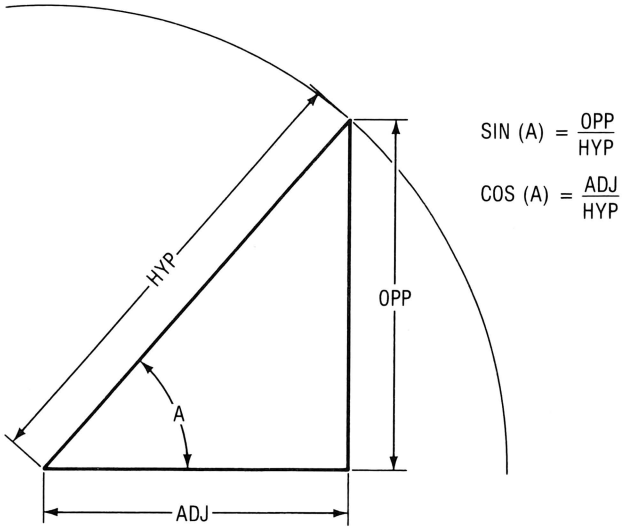


Figure 9.2 Demonstration of sine and cosine formulas.

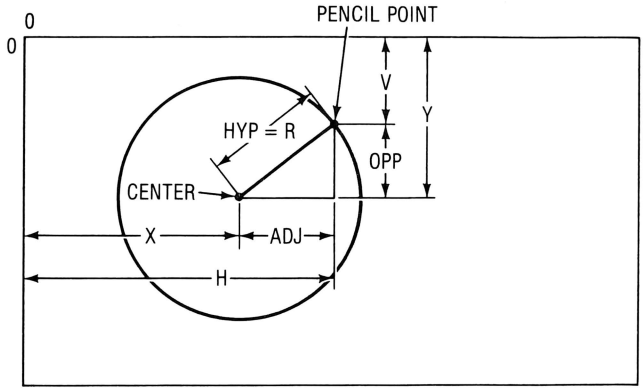


Figure 9.3 Computing horizontal distances *X* and *Y*.

Now SIN(A) = OPP/HYP = OPP/R, or, what is the same, OPP = R times SIN(A), and by similar reasoning, ADJ = R times COS(A). From the geometry of the situation shown above we must have

```
X = X + ADJ = X + R*COS(A)
Y = Y - OPP = Y - R*SIN(A)
```

The minus sign in the vertical component is there because we are mixing two different coordinate systems, one of them the traditional math system in which positive vertical units go up, the other the Applesoft BASIC system, in which positive vertical units go down.

Go ahead and enter and run this program. It will draw a circle of radius 25 with center at 100,75.

```
10  REM    ***CIRCLE***
15  REM
20  HGR : HCOLOR= 3: REM HCOLOR - WHITE
30 X = 100
40 Y = 75
50 R = 25
55 DA = .1
60  FOR A = 0 TO 2 * 3.14159 STEP DA
70 H = X - R *  COS (A)
80 V = Y - R *  SIN (A)
90  HPLOT H,V
100  NEXT
```

The variable DA in line 55 represents the amount by which to increase the angle (in radians) each time you plot a point.

Exercises

1. Enter different values of the angle increment DA. Run the program with values of DA = 1.0, 0.01 and 3.14159/2.

2. Modify the CIRCLE program so that it asks the user for the values of X, Y, and R, and also the number of points *N* around the circle. Calculate the angle increment DA by dividing twice PI by N. *Warning:* If the user inputs values that will cause the circle to go off screen, the program will halt with

Answer:

```
10  REM    ***EXERCISE 2***
20  REM
30  TEXT : HOME
40  REM
50  REM ----- INPUT DATA
60  REM
70  VTAB 5: PRINT "ENTER COORDINATES OF
    CENTER OF CIRCLE:"
80  VTAB 7: HTAB 8: INPUT "X = ";X$:X =
    VAL (X$): ON X = 1 OR X = 278 GOTO 80
90  VTAB 7: HTAB 20: INPUT "Y = ";Y$:Y =
    VAL (Y$): ON Y = 1 OR Y = 158 GOTO 90
100  VTAB 10: HTAB 2: INPUT "RADIUS =
     ";R$:R =  VAL (R$): ON R = 0 OR R =
     X OR R = Y OR R = 279 - X OR R =
     159 - Y GOTO 100
110  VTAB 14: HTAB 2: INPUT "NUMBER OF
     CIRCUMFERENCE POINTS = ";DA$:DA =
     VAL (DA$):
     ON DA = 1 GOTO 110:DA = 2 * 3.14159 /
      DA
120  REM
130  REM  ----- PROGRAM STARTS
140  REM
150  HGR : HCOLOR= 3: REM HCOLOR = WHITE
160  FOR A = 0 TO 2 * 3.14159 STEP DA
170  H = X - R *  COS (A)
180  V = Y - R *  SIN (A)
190  HPLOT H,V
200  NEXT
```

an error message. One solution is to test the values x and y just before HPLOTting them; if they go beyond the screen edge reset them to the edge. This can be done in four IF statements, one for each edge.

3. Write a program that draws random-sized circles at random locations in random colors with some random variation in the numbers of points per circle. The small circles require fewer points per circle. Try to develop a formula for the angle increment as a function of radius, one that draws a good circle without wasting time.

Answer:

```
10  REM   ***EXERCISE 3***
20  REM
30  TEXT : HOME : HGR
40  REM
50  REM ----- INPUT DATA
60  REM
80 X =  RND (1) * 278
90 Y =  RND (1) * 158
100 R =  RND (1) * (158 /
      2): ON R =  = 0 OR R = X OR
     R = Y OR R = 279 = X OR R = 159 -
      Y GOTO 100
110 DA = 2 * 3.14159 / ( RND (1) * 200)
120  REM
130  REM  ----- PROGRAM STARTS
140  REM
155  HCOLOR=  RND (1) * 6 -
      1: REM THIS WON'T GENERATE HCOLOR=0,
     (BLACK)!
160  FOR A = 0 TO 2 * 3.14159 STEP DA
170 H = X - R *  COS (A)
180 V = Y - R *  SIN (A)
190  HPLOT H,V
200  NEXT
300  GOTO 50
```

4. Draw a bow and arrow target with three concentric rings. The bull's-eye should be HCOLOR = 6, the next ring out should be HCOLOR = 2, and the outermost ring should have HCOLOR = 3. Make the target as large as will fit on the screen. The width of the outer rings should equal the diameter of the bull's-eye. *Hint:* Have a FOR . . . NEXT loop control the radius, making it move from center to edge. Three IF statements could set the color, depending on the radius.

Answer:

```
10  REM   ***EXERCISE 4***
20  REM
30  TEXT : HOME : HGR
40  REM
50  REM  ----- PROGRAM STARTS
60  REM
```

continued

```
70 RADIUS = (159 /
     2): GOTO 100: REM INITIALIZE RADIUS.
80 RADIUS = RADIUS - (159 /
    6): REM  INCREMENT RADIUS FOR
    (3) RINGS.
90  IF RADIUS = 0 THEN  HOME : VTAB 22:
      PRINT "YOU ARE NOW UNDER MY COMPLETE
      CONTROL,
      SO SEND ALL YOUR MONEY
      TO:";: VTAB 2: END
100  HCOLOR= (RADIUS - 26.5) * 6 -
          (RADIUS - 53) *
          2 - (RADIUS - 79.5) * 3: REM  THIS
          BOOLEAN FUNCTION EXPRESSION WILL
          MAKE
          HCOLOR EQUAL 6,2, OR3 AND DO IT IN
          ONE NESTABLE LINE INSTEAD
          OF (3) "IF" STATEMENTS!
110  IF RADIUS - 53 THEN  VTAB 22:
          PRINT  CHR$ (7)"NO THERE'S NOTHING
          WRONG WITH
          THE": PRINT" PLOTTING. THOSE
          LITTLE BLACK DOTS": PRINT" ARE
          ACTUALLY ARROW HOLES!!";: VTAB 1
120  FOR R = RADIUS TO RADIUS - (159 /
     6) STEP  -
     1: GOSUB 160: NEXT R: GOTO 80
130  REM
140  REM ----- CIRCLE PLOTTING SUBROUTINE.
150  REM
160  FOR A = 0 TO 3.14159 * 2 STEP 10 /
     (R - 20)
170 H = (279 / 2) - R *  COS (A)
180 V = (159 / 2) - R *  SIN (A)
190  IF A = 0 THEN  HPLOT H,V:HO = H:VO =
    V: NEXT A: GOTO 220
200  HPLOT  TO H,V
210  NEXT A
220  HPLOT  TO HO,VO: REM  THIS COMPLETES
    FULL CIRCLE!
230  RETURN
```

10. CREATING SHAPES

A feature of Applesoft BASIC that you will eventually want to take advantage of is its ability to display predefined shapes on the screen, after having scaled them and rotated them to your specifications. The definition of the shape does require some detail work, but once the shapes are in the main memory the rest is easy.

A *shape* is a string of unit vectors, each of which can point in one of four directions, up, down, right, left.

For example, a square of four pixels can be defined by the string of unit vectors in Figure 10.1. Assuming that we start in the upper left pixel:

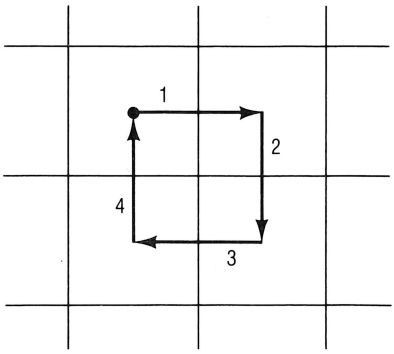


Figure 10.1 A string of unit vectors defines a square of four pixels.

There are four vectors in this shape, and all four call for the pixel at their tail to be plotted. The location of the tail of the first vector is the *origin* of the shape. It is the point about which the shape will be rotated if you use the ROTate command, and it is placed at the coordinates you specify when you use the DRAW or XDRAW commands.

For the machine to make use of your shape you will have to code the vectors into binary notation so that it can be stored in the machine's memory. This is accomplished by giving each unit vector a triple of *bits* $b_1\ b_2\ b_3$ where $b_1 = 1$ if the pixel from which you are moving is to be displayed, $b_1 = 0$ if the pixel is not to be displayed. The remaining pair $b_2\ b_3$ specifies the direction according to

Direction	b_2	b_3
up	0	0
down	1	0
left	1	1
right	0	1

Using these conventions we obtain the following triples:

Vector	b_1 b_2 b_3 Triple	
1	1 0 1	(plot and move right)
2	1 1 0	(plot and move down)
3	1 1 1	(plot and move left)
4	1 0 0	(plot and move up)

Now we are going to line these triples up into *bytes* (a byte is a string of eight bits) so that we can fit them into memory. For reasons you need not know at this point, we move from right to left, top to bottom, as we place two triples in each byte. The left-hand pair of bits in each byte we make into zeros. (It is possible to use the left-hand pair of bits for a move without a plot in any direction other than up. It is simpler, and sufficient, however, to just pad them with zeros.)

Since we pack two triples per byte, and we have four triples, it will take two bytes to code the shape. In addition, we will use one more byte, all zeros, to signal that the shape is complete. This produces

First byte:	0	0	1	1	0	1	0	1
Second byte:	0	0	1	0	0	1	1	1
Third byte:	0	0	0	0	0	0	0	0

Caution: If you make the 2nd triple of a byte a "move up without plotting," the move will not be performed. DRAW will ignore these bits if they are all zeros.

These are the actual bytes that will be in memory to represent our shape to the drawing routines.

To get these bytes into memory we need to convert them into hexadecimal (base 16), because this is what the monitor expects you to use when you are entering data directly into memory. This you do by taking four bits (called a *nibble*, or half-byte) and converting it to one character, a digit 0 through 9 or the letters A to F:

Binary	*Hex*
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	A
1011	B
1100	C
1101	D
1110	E
1111	F

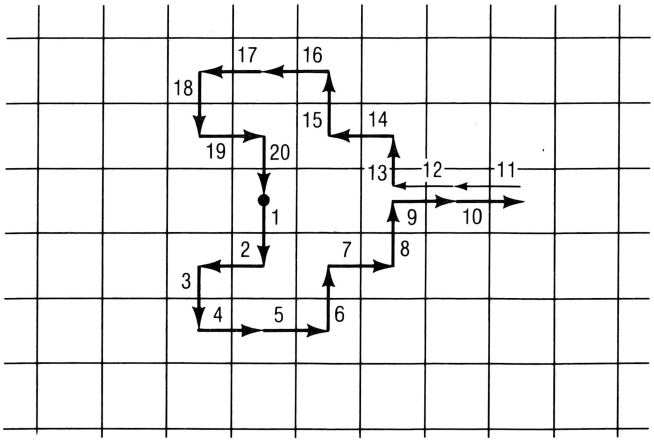
Each byte generates two hexadecimal characters:

First byte:	0	0	1	1	0	1	0	1 = 3	5
Second byte:	0	0	1	0	0	1	1	1 = 2	7
Third byte:	0	0	0	0	0	0	0	0 = 0	0

These three pairs of hexadecimal characters describe the shape. They are in a format that can be input into memory. This involves entering a shape table into memory, the subject of the next chapter.

We include one more example (see Figure 10.2) so that you can make sure you understand all the details.

1 110	5 101	9 101	13 100	17 111
2 111	6 100	10 101	14 111	18 110
3 110	7 101	11 011	15 100	19 101
4 101	8 100	12 011	16 111	20 110



1. 110	5. 101]	9. 101	13. 100	17. 111
2. 111	6. 100	10. 101	14. 111	18. 110
3. 110	7. 101	11. 011	15. 100	19. 101
4. 101	8. 100	12. 011	16. 111	20. 110

Figure 10.2 Spaceship shape #2.

The 20 vectors will fit into ten bytes. We will add one more byte of all zeros to signal the end of the shape. Our completed shape looks like this:

Byte	*Binary*	*Hex*
1	00111110	3 E
2	00101110	2 E
3	00100101	2 5
4	00100101	2 5
5	00101101	2 D
6	00011011	1 B
7	00111100	3 C
8	00111100	3 C
9	00110111	3 7
10	00110101	3 5
11	00000000	0 0

To enter this spaceship shape we must create a shape table and place it into memory. First get into the monitor by entering

```
CALL -151 (RETURN)
```

You should see an asterisk on the left. Then enter

```
decimal:  7676:    1  0  4  0 62 46 37 37 45
                  27 60 60 55 53  0
hex:      *1DFC:  01 00 04 00 3E 2E 25 25 2D
                  1B 3C 3C 37 35 00 (RETURN)
```

When the asterisk reappears, load the address of the shape table into location E8 by entering this:

```
decimal:  232:252  29
hex:      *E8:FC   1D (RETURN)
```

This tells the system that your shape table begins in location 1DFC. Finally you must get back to Applesoft, by pressing RESET.

Exercises

1. Check that your shape is properly entered into memory by getting into the monitor and entering *1DFC (RETURN). The contents of memory will be displayed in groups every time you press RETURN. Compare the display with what it should be.

2. Make sure your spaceship works by running this program

```
10 REM ***SPACESHIP TEST***
20 HGR:HCOLOR=3:SCALE=10:ROT=0
30 DRAW 1 at 100,75
```

11. USING SHAPES

Now that you have a spaceship shape safely stored in memory, it is time to learn how to display it. You might as well do some work in immediate execution mode, as this will be more interactive.

First enter HGR (without a line number) and press RETURN. Then enter HCOLOR=3 and press RETURN. Enter SCALE=1, press RETURN, then ROT=0, press RETURN. Now you can draw your spaceship anywhere you want on the screen.

For example, if you enter

```
DRAW 1 AT 130,75 (Press RETURN)
```

you should see the spacecraft (at a scale of 1) appear somewhere near the middle of the screen. When the SCALE is 1, each shape vector is exactly one high-resolution pixel long. If SCALE is set to some integer n ($n < 255$), then each vector in the shape *is repeated n times* before going on to the next vector. This is how the scaling is accomplished. Try displaying your shape at scales of 10, 20, 30, and so forth. Try moving the shape around on the screen.

The XDRAW command works the same way, except that it plots in the color ''opposite'' to what is already in the background. This is useful for several reasons. One is that the shape can be erased using a second XDRAW at the same location. Another is that XDRAW against an unknown background will always show up.

Inside the machine is a special register that holds the current value of ROT. A value of zero causes the shape to be drawn in the same orientation as the original shape. A value of 16 causes a 90° clockwise orientation; 32 causes 180°; 48 causes 270°; and 64 causes 360°, or what is the same, the original. Values greater than 64 act as if 64 had been subtracted first. (They are taken ''modulo 64.'')

Using ROTATE you can make some nice radially symmetric designs using any shape. For example, try this one:

```
10  REM  ***DESIGN 2***
12  REM
13  REM ----- POKE "SPACESHIP" INTO MEMORY.
14  REM
15  FOR P = 7676 TO 7676 - 14: READ D:
        POKE P,D: NEXT P: POKE 232,252:
        POKE 233,29:
        DATA 1,0,4,0,62,46,37,37,45,27,60,
        60, 55,53,0: REM
16  REM
17  REM ----- PROGRAM STARTS
18  REM
20  HGR
30  FOR I = 1 TO 20
35  SCALE= 5 * I
```

continued

```
40  FOR R = 0 TO 62 STEP 4
50  ROT= R
60  HCOLOR=  INT ( RND (1) * 7)
70  DRAW 1 AT 130,75
80  NEXT : NEXT
```

Exercises

1. Enter SHAPE #2 (spaceship) into memory and then draw it, at a SCALE of 10, ROTation of 8.

2. Enter this short program:

```
10 HGR: HCOLOR=3
20 INPUT S: SCALE=S: DRAW 1 AT
   100,75
30 GOTO 20
```

   This will let you see the spaceship at any scale you enter.

Answer:

```
1  REM    ***EXERCISE 2***
2  REM
3  REM ----- POKE "SPACESHIP" INTO MEMORY.
4  REM
5  FOR P = 7676 TO 7676 - 14: READ D:
        POKE P,D: NEXT P: POKE 232,252:
        POKE 233,29:
        DATA 1,0,4,0,62,46,37,37,45,27,60,
        60,55, 53,0: REM
6  REM
7  REM ----- PROGRAM STARTS
8  REM
10  HGR : HCOLOR= 3
20  INPUT "SCALE = ";S: SCALE= S: DRAW 1 AT
     100,75
30  GOTO 20
```

3. Write a short program to display the spaceship at SCALE = 30 for rotations 8 through 24, spaceship center at 130,80.

Answer:

```
1  REM    ***EXERCISE 3***
2  REM
3  REM ----- POKE "SPACESHIP" INTO MEMORY.
4  REM
5  FOR P = 7676 TO 7676 - 14: READ D:
        POKE P,D: NEXT P: POKE 232,252:
        POKE 233,29:
        DATA 1,0,4,0,62,46,37,37,45,27,60,
        60, 55, 53,0: REM
6  REM
7  REM ----- PROGRAM STARTS
8  REM
10  HGR : HCOLOR= 3
20  SCALE= 30
30  FOR X1 = 8 TO 24: ROT= X1
40  DRAW 1 AT 130,80
50  NEXT X1
```

4. Write a program to control the horizontal position of the spaceship (SCALE = 10, ROT = 0) which starts at 100,75. Get a single character from the keyboard. If it is not 1 or 2 get another character. If it is 1 or 2, first erase the ship, by XDRAW at the current location, then update the current location (if 1 subtract 1 from H, if 2 add 1 to H). Then XDRAW in the current location.

Answer:

```
10  REM  ***EXERCISE 4***
20  REM
30  REM ----- POKE "SPACESHIP" INTO MEMORY.
40  REM
50  FOR P = 7676 TO 7676 - 14: READ D:
        POKE P,D: NEXT P: POKE 232,252:
        POKE 233,29:
        DATA 1,0,4,0,62,46,37,37,45,27,60,
        60, 55, 53,0: REM
60  REM
70  REM ----- PROGRAM STARTS
80  REM
90  TEXT : HOME : HGR
100  HCOLOR= 3: SCALE= 10: ROT= 0
110 H = 100:V = 75: DRAW 1 AT H,V: REM
    "SPACESHIP" STARTING POSITION.
120  VTAB 22: HTAB 1: PRINT "ENTER -1> OR -
    2> FOR LEFT/RIGHT :";: GET A$:A
    =  VAL (A$): IF A = 1 OR A = 2
    THEN 120
130  XDRAW 1 AT H,V
140  IF A = 1 THEN H = H - 1
150  IF A = 2 THEN H = H - 1
160  DRAW 1 AT H,V
170  GOTO 120
```

5. Add some steps to your program from exercise 4 above to make it rotate left (L) or right (R). You can start the ROT value at 64, subtract 1 if L, add 1 if R. If the value of ROT becomes less than zero you will get an error message. You could test it before you decrement, to avoid this message.

Answer:

```
10  REM  ***EXERCISE 5***
20  REM
30  REM ----- POKE "SPACESHIP" INTO MEMORY.
40  REM
50  FOR P = 7676 TO 7676 - 14: READ D: POKE
        P,D: NEXT P: POKE 232,252:
        POKE 233,29:
        DATA 1,0,4,0,62,46,37,37,45, 27,
        60,60, 55, 53,0: REM
60  REM
70  REM ----- PROGRAM STARTS
80  REM
90  TEXT : HOME : HGR
100  HCOLOR= 3: SCALE= 10: ROT= 64
110 H = 100:V = 75: DRAW 1 AT H,V: REM
    "SPACESHIP" STARTING POSITION.
```

cont.

```
120  VTAB 22: HTAB 1: PRINT "ENTER -1> OR -
     2> FOR LEFT/
     RIGHT :";: GET A$: ON A$ = "1" OR A$
     = "2" OR A$ = "L" OR A$ = "R" GOTO
     130: GOTO 120
130  REM
140  XDRAW 1 AT H,V: REM THERE SEEMS TO BE
          & ERROR IN THE 'XDRAW' ROUTINE!
          IF YOU CHANGE COLOR AND RE'DRAW'
          IN BLACK (0) IT DOES ERASE!
150  IF A$ = "1" THEN H = H - 1:
     REM MOVE LEFT.
160  IF A$ = "2" THEN H = H -
      1: REM MOVE RIGHT.
170  IF A$ = "R" THEN RR = RR -
      1: REM ROTATE RIGHT.
180  IF A$ = "L" AND RR = 0 THEN RR = RR -
      1: REM  ROTATE LEFT.
190  ROT= RR
200  DRAW 1 AT H,V
210  GOTO 120
```

6. Add two more features to your program from exercise 5 above and call it ***SPACESHIP CONTROL***. If U move it up, if D move it down, if B make it bigger, if S make it smaller. Debug your program so that it keeps running.

```
100  REM    ***SPACESHIP CONTROL***
110  REM
120  REM -----
      POKE "SPACESHIP" INTO MEMORY.
130  REM
140  FOR P = 7676 TO 7676 - 14: READ D:
          POKE P,D: NEXT P: POKE 232,252:
          POKE 233,29:
          DATA 1,0,4,0,62,46,37,37,45,27,
          60, 60, 55, 53,0: REM
150  REM
160  REM ----- PROGRAM STARTS
170  REM
180  TEXT : HOME : HGR
190  HCOLOR= 3:SS = 10: SCALE= SS:RR = 64:
     ROT= RR
200 H = 100:V = 75: DRAW 1 AT H,V: REM
     "SPACESHIP" STARTING  POSITION.
210  VTAB 22: HTAB 1: PRINT "<1>LEFT
            <R> ROT    <U>P     <B>IGGER":
          PRINT "<2>RIGHT  <L> ROT
          <D>OWN    <S>MALLER": VTAB 1:
          GET A$
220  REM
230  REM ----- PLOTTING MOVES.
240  REM
250  XDRAW 1 AT H,V: REM THERE SEEMS TO BE
          & ERROR IN THE 'XDRAW' ROUTINE!
          IF YOU CHANGE COLOR AND RE'DRAW'
          IN BLACK (0) IT DOES ERASE!
260  IF A$ = "1" THEN H = H - 1: GOTO
     350: REM MOVE LEFT.
```

```
270  IF A$ = "2" THEN H = H -
      1: GOTO 350: REM MOVE RIGHT.
280  IF A$ = "R" THEN RR = RR -
      1: GOTO 350: REM ROTATE RIGHT.
290  IF A$ = "L" AND RR = 0 THEN RR = RR -
      1: GOTO 350: REM  ROTATE LEFT.
300  IF A$ = "U" THEN V = V -
      1: GOTO 350: REM MOVE UP, OR "GET-IT-
     UP"!
310  IF A$ = "D" THEN V = V - 1: GOTO 350:
     REM MOVE IT DOWN, OR "GET DOWN"!
320  IF A$ = "B" AND SS = 255 THEN SS =
     SS - 1: GOTO 350: REM MAKE IT
     BIGGER???
330  IF A$ = "S" AND SS = 1 THEN SS = SS -
      1: GOTO 350: REM LET THE AIR OUT. . .

340  GOTO 210: REM PICK A VALID KEY STUPID!
350  ROT= RR
360  SCALE= SS
370  DRAW 1 AT H,V
380  GOTO 210
```

7. Add one more feature to ***SPACESHIP CONTROL*** that will amaze your friends. Have it fire a missile when you enter the letter F by jumping to a subroutine that draws the missile track. Seek competent help with this problem.

12. TIC-TAC-TOE

The time has come for you to put together what you have learned so far. For this purpose the simple game of tic-tac-toe provides a good project for you to work on, since it requires all the features that we have discussed.

The idea is to create two shapes, an X and an O, and put them into a shape table. Thereafter you can scale them larger or smaller to suit your idea of a good-looking game screen. In the program developed here we will make our X with a cross, and then rotate it 45 degrees. This is easier than drawing it in its usual position. Similarly we will use the square of the preceding section as our O.

The cross is obtained with seven vectors:

```
RIGHT PLOT        1  0  1
LEFT NO PLOT      0  1  1
UP PLOT           1  0  0
DOWN NO PLOT      0  1  0
LEFT PLOT         1  1  1
RIGHT NO PLOT     0  0  1
DOWN PLOT         1  1  0
```

When packed into bytes at two vectors per byte these instructions become

```
BYTE 1    0  0  0  1  1  1  0  1
BYTE 2    0  0  0  1  0  1  0  0
BYTE 3    0  0  0  0  1  1  1  1
BYTE 4    0  0  0  0  0  1  1  0
BYTE 5    0  0  0  0  0  0  0  0
```

The required string of hexadecimal pairs that represent this shape is therefore

```
1D 14 0F 06 00
```

Recall that the string of hexadecimal pairs that represents the small square is

```
35 37 00
```

Using these two shapes we will next examine the general form of a shape table. The elements that must be present are explained on the right of the table.

Our shape table looks like this:

```
BYTE 1    02  (number of shapes)
BYTE 2    00  (always zero)
BYTE 3    06  (least significant byte)
BYTE 4    00  (most significant byte)
BYTE 5    0B  (index to first byte of shape 2)
BYTE 6    00
BYTE 7    1D  (first byte shape 1)
BYTE 8    14
BYTE 9    0F
BYTE 10   06
BYTE 11   00
BYTE 12   35  (first byte shape 2)
BYTE 13   27
BYTE 14   00  (last byte in table)
```

The offset addresses of each of the shapes in the table are taken relative to the starting address of the table. For example, consider shape 2. The contents of BYTE 5 are in hexadecimal. This gives an offset of 11, so that shape 2 will start in BYTE 12.

Now to enter this shape table into the machine's memory recall that you first get into the monitor by entering

```
CALL -151 (RETURN)
```

and then enter, after the asterisk

```
*1DFC: 02 00 06 00 0B 00 1D 14 0F
06 00 35 27 00
```

Now your shapes are in memory. Finally you tell Applesoft where your shape table starts, in this case by entering

```
*E8:FC 1D (RETURN)
```

Before you can draw the shapes you must get back into Applesoft BASIC by entering

```
*FP (RETURN)
```

When you see the Applesoft prompt character appear enter

```
SCALE=25:ROT=B (RETURN)
```

This will prepare the system to draw the cross so that it looks like an X, and will be the right size. Now enter high-resolution graphics and set the plotting color to white:

```
HGR: HCOLOR=3 (RETURN)
```

You are ready to draw. Enter

```
DRAW 1 AT 75,75 (RETURN)
```

You should see an X appear near the center, about five print lines high. If you don't get the X you need to debug either your program or the shape table. Remember that when you enter the monitor your program will be lost. Therefore save your program before you enter the monitor, so that you can restore it after you have checked that the shape table contains what it should, and that the system knows its correct starting address.

A very common error in high-resolution graphics is to forget to put the H in front of COLOR. If you get no display at all, that is the first thing to check.

To test the O, enter

```
SCALE=30: ROT=0 (RETURN)
DRAW 2 AT 59,59 (RETURN)
```

You should get an O superimposed over the X. If you don't get it, debug your program until you do.

If your X and O are working properly you can start the program described below. Do not turn off the

power to the Apple from this point on, as the shape table disappears when the power does. Pressing RESET is OK any time.

The first few lines of the program will draw the two horizontal guidelines and the two vertical guidelines, just as you do when you play the game on paper.

```
10  REM    ***TIC TAC TOE****
14  REM
15  FOR P = 7676 TO 7676 - 13: READ D:
        POKE P,D: NEXT P:
        POKE 232,252: POKE 233,29:
        DATA 2,0,6,0,11,0,29,20,15,6,0, 53,
        39, 0: REM  POKE "+" & "O" INTO
        SHAPE TABLE.
16  REM
20  PRINT "WELCOME TO TIC TAC TOE"
30  HGR : HCOLOR= 3
40  HPLOT 50,0 TO 50,150
50  HPLOT 100,0 TO 100,150
60  HPLOT 0,50 TO 150,50
70  HPLOT 0,100 TO 150,100
```

Go ahead and enter this much and run it. You should get the four lines on the screen (see Figure 12.1). Press RESET to get back to the program. This will spare you having to write TEXT in order to have the entire screen available to see your program.

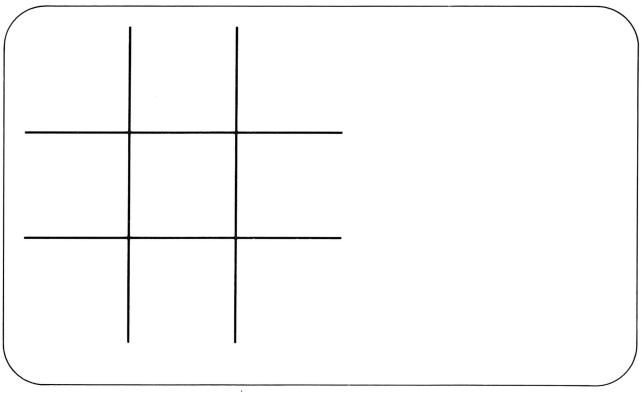


Figure 12.1 Tic tac toe grid

Next we find out from the user what symbol is to be displayed, and where it goes.

```
80  INPUT "WHOSE MOVE (X OR O)? ";M$
90  INPUT "WHICH COLUMN (1 TO 3)? ";H
100  INPUT "WHICH ROW (1 TO 3)?";V
```

Go ahead and run this much, to make sure the input statements work. It is always much better to debug each piece as you develop it, because if you find an error, and you know that the part just before works, the error will be in the new part. This cuts the error search time down considerably.

The last part of the program draws either an X or an O at the spot the user indicates.

```
110  IF M$ = "X" THEN  SCALE= 25:
     ROT= 8: DRAW 1 AT 50 * H - 25,50 * V - 25
```

```
120  IF M$ = "O" THEN  SCALE= 30: ROT= 0:
     DRAW 2 AT 50 * H - 41,50 * V - 41
130  GOTO 80
```

This is the entire program. The amounts to subtract from the horizontal and vertical coordinates in lines 110 and 120 were determined by experiment, as were the scale settings. If a lot of such experimenting is required it would be better to spread 110, say, into three more lines with statement numbers, so that one parameter can be changed at a time.

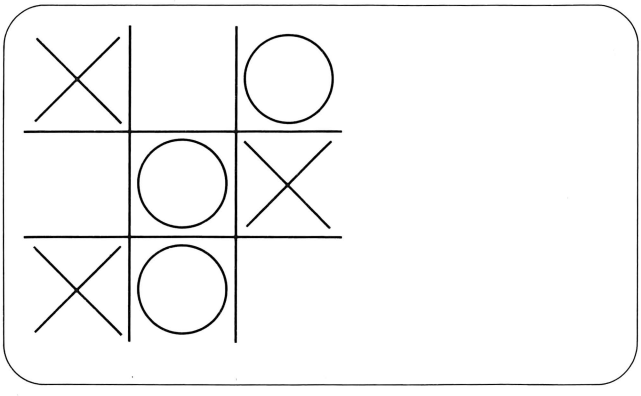


Figure 12.2

Exercises

1. Revise the program so that the O is bigger. Make it so big that it creates some tension by getting close to the guidelines. Keep it centered.

2. Revise the program by tilting the X one rotation unit to the left. Can you notice the tilt?

3. Revise the shapes for X and O so that they are more interesting and elaborate, with many more vectors. Try to get them to look like a commercial program. This exercise will let you know if you really want to be a video game programmer.

13. WHERE TO GO FROM HERE

Now that you have mastered some of the features of Applesoft BASIC, you should have some idea if you want to continue with more graphics on the Apple. If you do, there is a wealth of software, hardware, manuals, and texts available to you.

You will want to invest in some graphics software, of which there is plenty. Consult *The Book of Apple Software 1983* or later for a look at what is available. If you see a graphics package that you like on another microcomputer, you probably can find a version for the Apple. You can get many different types of graphic input to your Apple, including several graphics tablets and slow-scan television camera systems. Any device that lets you gather graphic input from the real world will prove a good investment.

One useful piece of software for high-resolution graphics is a system that will transfer your filled-up high-resolution screen to a dot matrix printer like the Epson. You cannot get color, but you get a good grey scale rendition of what you see on the screen.

If you enjoy computer artwork in color, one good approach is to photograph the color TV or monitor screen with a 35mm slide camera. You can make prints from it of any size, with a final resolution much better than the screen itself. If you want to submit designs for publication, the 35mm slide is a good format. Before you start work in this area, however, look at the low-resolution designs of Alan Foxx and Todd Holforty.

There are two basic directions you can go from here: more programming or less programming. If you opt for more programming, then learning assembly language is your next step. The book by Jeffrey Stanton on arcade graphics makes a good guide in this area. The more machine language you know, the more you will appreciate the Apple. It is really a very powerful machine.

If you want to do less programming and more graphics, find the best software packages available and start working with them. Many of them require a graphic tablet for input, or at least a joystick. You should have a graphic tablet of some sort if you plan to do much work in this area, as a keyboard is not the ideal interface for pictorial material.

Bibliography

1. Apple Computer, Inc., *The Applesoft Tutorial.*
2. Apple Computer, Inc., *Applesoft II BASIC Programming Reference Manual.*
3. Apple Computer, Inc., *The DOS Manual.*
4. Alan Foxx and Todd Holforty, "Low-Res Fantasy Art," *Creative Computing,* July 1983.
5. Richard G. Peddicord, *Understanding Apple BASIC.* Alfred Publishing Co., 1983.
6. Jeffrey Stanton, *Apple Graphics & Arcade Game Design.* The Book Company.
7. Jeffrey Stanton and Robert P. Wells, *The Book of Apple Software 1983.* The Book Company.

Glossary

angle The amount of rotation between two intersecting lines. Measures of angle include degrees, radians, and ROTation units.

array A collection of storage cells organized into a one-dimensional vector or a two-dimensional matrix.

arrow keys Special keys for moving across displayed characters without changing them.

bit 0 or 1, off or on.

brightness The amount of light given off by a visible object.

byte A string of eight bits.

CALL −151 Used to get into the monitor from Applesoft.

circle A collection of points equally distant from a fixed point, the center.

COLOR= Sets one of 16 colors in low resolution.

column A collection of storage cells arranged in a vertical tower.

contrast A measure of the range in brightness in a displayed scene.

coordinates The pair of values that specify where a picture element belongs.

COS function The cosine function.

CRT *C*athode *R*ay *T*ube, TV screen.

cursor The flashing square that lets you know where your next character will be displayed.

DRAW command Used in high resolution to draw a preprogrammed shape.

flag designs Designs in low resolution based on logical constraints.

floating-point arithmetic Nine-digit accuracy arithmetic done by Applesoft BASIC.

floating-point numbers Nine-digit accuracy numbers used by Applesoft BASIC.

HCOLOR= command Selects one of eight colors for high-resolution plotting.

hexadecimal A number system based on 16 symbols.

HGR1 command Causes system to enter high-resolution mode using screen 1. RAM locations, 2000.3FFF.

HGR2 command Causes system to enter high-resolution mode using screen 2. RAM locations, 4000.5FFF.

high resolution The 280 by 160 (or 192) high-resolution system of the Apple //.

HLIN command Draws a horizontal line in low resolution.

horizontal Going from left to right.

HPLOT command Plots a pixel in high resolution.

hue The apparent color of an object.

ideal line The make-believe line between two points that we approximate with fixed-resolution display systems.

INTeger function Chops off decimals.

kaleidoscope A tubelike device with mirrors that generates random patterns with high radial symmetry.

line A collection of points that maintain the same bearing between two selected points.

low resolution The 40 by 40 low-resolution system of the Apple //.

major diagonal The diagonal of a square that goes from upper left to lower right.

minor diagonal The diagonal of a square that goes from lower left to upper right.

monitor TV screen, high resolution video display, CRT; or the operating system of the Apple //.

picture element The smallest visual object that a display system can generate.

pixel Same as a picture element.

radian A measure of angle. Twice pi radians equals 360 degrees.

reflecting line An imaginary line used in our computer kaleidoscope.

resolution A measure of the amount of information on a visual screen.

RND function Produces a new random number between 0 and 1 each time it is called.

ROT= command Sets the amount of clockwise rotation for a shape before it is drawn.

SCALE= command Sets the number of repetitions of each unit vector as the shape is DRAWn. This scales the shape.

shape A string of unit vectors that define a shape in high resolution.

shape table Several shape strings in a row, with pointer information at the top.

SIN function The sine function.

slope The ratio of vertical distance traveled to horizontal distance traveled along a straight line.

unit vector The individual triple of binary bits that defines one vector in the shape.

vertical Going up or down.

NOTES

NOTES

NOTES

NOTES

NOTES

GET ON THE ALFRED COMPUTER MAILING LIST!

KEEP UP-TO-DATE!

Send us your complete **name** and **address,** and we'll send you catalogs, newsletters, and new product listings, as they become available.

Or fill out and mail this coupon:

Name _____

Address _____

City _____ State _____ Zip _____

Handy Guide Titles You Own _____

Comments: _____

Send to: ALFRED PUBLISHING CO., INC.
Post Office Box 5964
Sherman Oaks, California 91413

Alfred Handy Guides

Practical, economical, and concise
Alfred Handy Guides tell you what you need to know quickly and easily — without a lot of reading!

"A busy executive or professional who feels the need for a crash course in the subjects covered by the **Handy Guides** *would do well to pick up a few on the way to the commuter train or the airport."*
Personal Computing Magazine

The Alfred Handy Guide Series to Computers

How to Buy a Personal Computer
How to Buy a Portable Computer
How to Buy a Word Processor
How to Use Atari Computers
How to Use the Commodore 64
How to Use the IBM PC
How to Use VisiCalc/SuperCalc
The Personal Computer Glossary
Understanding APL
Understanding Apple Basic
Understanding Apple Graphics
Understanding Artificial Intelligence
Understanding Atari Graphics
Understanding BASIC
Understanding COBOL
Understanding Data Base Management
Understanding Data Communications
Understanding Electronic Mail
Understanding FORTH
Understanding FORTRAN
Understanding LISP
Understanding LOGO
Understanding Pascal
Understanding Pilot
Understanding Robots

Look for new titles and new series.
For more information:

Alfred Publishing Co., Inc.
15335 Morrison St.
P.O.Box 5964
Sherman Oaks, CA 91413

ISBN 0-88284-250-1

0 38081 00295 00250